DOCUMENT RESUME

ED 038 313 SE 008 276

AUTHOR	McCormick, William T., Jr.; And Others
TITLE	Identification of Data Structures and Relationships by Matrix Reordering Techniques.
INSTITUTION	Institute for Defense Analysis, Arlington, Va.
REPORT NO	RP-512
PUB DATE	Dec 69
NOTE	141p.
EDRS PRICE	EDRS Price MF-$0.75 HC-$7.15
DESCRIPTORS	*Algorithms, *Data Analysis, Data Processing, Mathematical Applications, Mathematical Models, *Mathematics, *Research Methodology, *Statistical Analysis

ABSTRACT

Presented are the results of a study conducted to develop algorithms for ordering and organizing data that can be presented in a two-dimensional matrix form. The purpose of the work was to develop methods to extract latent data patterns, grouping, and structural relationships which are not apparent from the raw matrix data. The algorithms developed are (1) Moment Ordering Algorithm, (2) Moment Compression Algorithm, and (3) Bond Energy Algorithm. They are applicable to a variety of problems involving multivariate data analysis. (RS)

ED0 38313

F-DOD

CER

SE

U.S. DEPARTMENT OF HEALTH, EDUCATION & WELFARE
OFFICE OF EDUCATION

THIS DOCUMENT HAS BEEN REPRODUCED EXACTLY AS RECEIVED FROM THE PERSON OR ORGANIZATION ORIGINATING IT. POINTS OF VIEW OR OPINIONS STATED DO NOT NECESSARILY REPRESENT OFFICIAL OFFICE OF EDUCATION POSITION OR POLICY.

RESEARCH PAPER P-512

IDENTIFICATION OF DATA STRUCTURES AND RELATIONSHIPS BY MATRIX REORDERING TECHNIQUES

William T. McCormick, Jr.
Stephen B. Deutsch
John J. Martin
Paul J. Schweitzer

December 1969

IDA
INSTITUTE FOR DEFE
SYSTEMS EVALUATION

IDA
INSTITUTE FOR DEFENSE ANALYSES
400 ARMY-NAVY DRIVE
ARLINGTON, VIRGINIA 22202

SE 008 276

IDA Log No. HQ 69-10829
Copy 78 of 150 copies

The information contained in this publication was developed under the IDA's independent research program. Its publication by IDA does not imply endorsement by the Department of Defense or other Government agency nor should the contents be construed as reflecting the official position of any U.S. Government office.

ED038313

APR 1 1970

RESEARCH PAPER P-512

IDENTIFICATION OF DATA STRUCTURES AND RELATIONSHIPS BY MATRIX REORDERING TECHNIQUES

William T. McCormick, Jr.
Stephen B. Deutsch
John J. Martin
Paul J. Schweitzer

December 1969

IDA
INSTITUTE FOR DEFENSE ANALYSES
SYSTEMS EVALUATION DIVISION
400 Army-Navy Drive, Arlington, Virginia 22202

FOREWORD

This paper presents the results of a study undertaken to develop methods for ordering and organizing technical, social, economic and other data that can be presented in array form. The study leading to the development of this report was conducted as independent research at the Institute for Defense Analyses. The theory and development of the algorithms described in this paper are the work of members of the Systems Evaluation Division.

ABSTRACT

This research paper presents the results of a study conducted to develop algorithms for ordering and organizing data that can be presented in a two-dimensional matrix form. The only restriction imposed on the analysis was that the rows and columns of the raw input data matrices could only be reordered, thus preventing the creation of artificial coefficients or loss of essential input information. The purpose of this work was to develop methods to extract latent data patterns, groupings, and structural relationships which are not, in general, apparent from the raw matrix data.

Three distinct algorithms were developed and are presented in detail within the report. They have been applied to a variety of examples from the social and technical sciences which will also be discussed. The first method developed, the *Moment Ordering Algorithm,* has proven to be an effective technique for uncovering and displaying a dominant univariate relationship between the two sets of entities that lie along the vertical and horizontal axes of a matrix. The second method, the *Moment Compression Algorithm,* is designed to factor decomposible matrices by proper reordering but was not applied extensively because of its complex and time-consuming solution. The last method developed, the *Bond Energy Algorithm,* was found to be applicable to a broader class of problems than the first two methods and is able to efficiently organize, group, and interrelate data of considerably more complex structure.

It will be shown that the techniques developed in this work are applicable to a variety of problems involving multivariate data analysis and, when used, can often significantly augment the level of understanding and comprehension of complicated multivariate relationships.

CONTENTS

PART I: GENERAL DESCRIPTION 1

I. INTRODUCTION 3

II. DESCRIPTION AND OBJECTIVES OF THE THREE ALGORITHMS 5
- A. The Bond Energy Algorithm 5
- B. The Moment Ordering Algorithm 8
- C. The Moment Compression Algorithm 9
- D. Contrasts Among the Three Algorithms 10

III. CONCLUSIONS 13

PART II: DETAILED DESCRIPTIONS AND APPLICATIONS 15

I. THE BOND ENERGY ALGORITHM 17
- A. Motivation 17
- B. The Measure of Effectiveness 18
- C. The Solution: Maximization of the ME 21
- D. Operation of the Algorithm 24
- E. Applications 26
- F. Examples 27

II. THE MOMENT ORDERING ALGORITHM 49
- A. Introduction 49
- B. The Algorithm 49
- C. Results 55

III. THE MOMENT COMPRESSION ALGORITHM 63
- A. Introduction 63
- B. Measure of Effectiveness for Moment Compression 65
- C. Gradient Algorithm for Approximate ME-Optimization 66
- D. Computational Results 68

REFERENCES 75

APPENDIXES

APPENDIX A Formulation of the Bond Energy ME Optimization as Two Quadratic Assignment Problems 79

APPENDIX B Proof that the Bond Energy Suboptimal Algorithm Will Produce Block Factored Form if it is Possible to do so by Row and Column Permutations 83

APPENDIX C Proof that the Moment Compression Algorithm Will Produce Block Factored Form if it is Possible to do so by Row and Column Permutations 89

APPENDIX D The Bond Energy Computer Program 95

APPENDIX E The Moment Ordering Computer Program 107

APPENDIX F The Moment Compression Computer Program 123

APPENDIX G Discussion of Measures of Effectiveness 139

APPENDIX H A Measure of Effectiveness for the Moment Ordering Algorithm 145

APPENDIX I Multiple Solutions to the Moment Ordering Algorithm for a Sample 3 x 3 Array 149

FIGURES

1. Representation of Bond Energy ME 6
2. Illustration of the Sensitivity of the Bond Energy ME 7
3. An Example of the Moment Ordering Algorithm 9
4. Matrix With Perfect Block Form 10
5. Relationship Matrix Showing 4 Unique Groups 17
6. Relationship Matrix With Block-Checkerboard Form 18
7. Initial Non-Binary Similarity Matrix 23
8. Reordered Non-Binary Similarity Matrix 23
9. Initial Binary Object-Attribute Matrix 27
10. Initial Fractional Object Similarity Matrix 28
11. Initial Binary Similarity Matrix 28
12. Reordered Binary Object-Attribute Matrix 29
13. Reordered Fractional Object Similarity Matrix 29
14. Reordered Binary Similarity Matrix 30
15. Initial Matrix of Marketing Techniques and Applications 31
16. Reordered Matrix of Marketing Techniques and Applications 33
17. Dependency Matrix for Airport Variables 39
18. Reordered Dependency Matrix for Maximum Clumpiness 39
19. Suggested Subproblems With Coordination Links 40
20. Reordered (Hindi) Consonant Error Matrix 42
21. Initial Inter-City Inverse Distance Matrix 44
22. Reordered Inverse Distance Matrix 44
23. Geographical Illustration of City Clusters 45
24. Operation of the Algorithm on a Small Array 52
25. Illustration of Multiple Solutions 53
26. Illustration of Cycling Phenomenon 54
27. Initial Word Relationship Array, Matrix A-1 69
28. Reordered Word Relationship Arrays 70
29. Initial Similarity Matrix B-1 72
30. Reordered Similarity Matrices 74

C-1 Sample Matrix 92
D-1 Flow Chart for Sequential Row (or Column) Selection and Laydown for Bond Energy Algorithm 98
E-1 Flow Chart for Moment Ordering Algorithm 110
E-2 Sample Data Deck for Card Input Version of Moment Ordering Algorithm 111
F-1 Flow Chart for Moment Compression Algorithm 127
I-1 Experimental 3 by 3 Array 151
I-2 Multiplicity of Solutions for a Small Array 152

TABLES

1. Bond Energy Algorithm Computation Time 25
2. Frequency Distribution of ME 26
3. Airport Variables 37
4. Senate Roll Call Votes Included in Analysis 56
5. Arrays Used in Senate Vote Pattern Analysis 57
6. Results of Vote Pattern Analysis 58
7. Raw Pottery Percentages 59
8. Agreement Coefficients 60
9. Reordered Agreement Coefficients 61

PART I: GENERAL DESCRIPTION

I. INTRODUCTION

Since the introduction of the large digital computers, methods of multivariate analysis[1] are being developed that utilize more effectively the computational resources and characteristics of the computer than some of the more conventional and established statistical techniques. These new methods are being employed because it is now possible to undertake data analysis problems in considerably greater detail than was previously feasible. A class of techniques that is able to account for detailed individual relationships as well as macroscopic data structure is exemplified by the cluster-seeking[2] methods. Ball (Ref. 1) has accurately pointed out that many classical statistical techniques depend heavily on statistical quantities estimated from the data and that this "averaging" from the data can sometimes lead to erroneous conclusions. This is simply because microscopic variations in the data cannot, in general, be detected from the statistical quantities estimated with the result that small but significant information can be overwhelmed and even lost under the pressure of larger statistical trends. Furthermore, many of these classical techniques such as principal component analysis (Ref. 28) or factor analysis (Ref. 28) implicitly assume data distributions that are not always present. Thus, it appears that there is a definite need for better direct analysis techniques so that it is not necessary to completely rely on functions of data or on assumptions regarding their distribution.

This paper presents three new direct data analysis techniques that were developed at the Institute for Defense Analyses. One of the algorithms, the *Bond Energy Algorithm*, shares a few of the same objectives as some of the other cluster-seeking techniques (Refs. 2 through 20) but has several important differences and advantages. The *Moment Ordering Algorithm* has as its principal goal the discovery of a single dominant relationship in the data, while the *Moment Compression Algorithm* attempts to factor the data into separable pieces or clusters. Two important characteristics that all three of these methods share is that they operate directly on the non-negative raw input matrix data and that they reorganize and reorder the matrix data by performing row and column permutations in order to reveal obscure and potentially informative data patterns. The output of all these algorithms, then, is a new data matrix with its resulting new ordering.

---

[1] Multivariate Analysis includes such mathematical techniques as Regression Analysis, Factor Analysis, Principal Component Analysis, Canonical Analysis, Cluster Analysis, etc.

[2] Cluster Seeking techniques are those data analysis methods which seek to identify groups of similar entities.

In Chapter II, the most important features and characteristics of each of the three algorithms will be briefly described. Then, in Chapter III, the major results and conclusions of this study will be presented. Part II of this paper contains a detailed description of the theory and development of the three algorithms along with a number of pertinent examples which illustrate the favorable characteristics and general applicability of these methods.

II. DESCRIPTION AND OBJECTIVES OF THE THREE ALGORITHMS

In this chapter the three data ordering algorithms are briefly described and their objectives are compared. More detailed description of the theory and development of these algorithms, along with a number of applications, will be found in Part II.

A. THE BOND ENERGY ALGORITHM[1]

The Bond Energy Algorithm[2] is capable of identifying and displaying natural groups and clusters that occur in complex data matrices. Moreover, the algorithm is able to uncover and display the associations and interrelationships of these groups with one another. These tasks are accomplished through the use of a numerical measure of how clustered or clumpy[3] a matrix is. The proposed measure of effectiveness (ME) attains its maximum value when the matrix assumes a very clumpy or aggregated form. It has been found that the structures and relationships existing in data matrices more clearly exhibit themselves when the matrices are presented in more aggregated forms corresponding to larger MEs.

The ME is defined as follows. Assume that the matrix of relationships (or transactions, flow, etc.) has dimension M by N with non-negative elements a_{ij}. The quantity α_{ij} is defined as[4]

$$\alpha_{ij} = \frac{1}{2}\left[a_{i+1,j} + a_{i-1,j} + a_{i,j+1} + a_{i,j-1}\right].$$

From Fig. 1 it can be seen that α_{ij} is just one half the sum of the horizontal and vertical nearest neighbors of a_{ij}. The unnormalized ME can now be defined as

$$\text{ME} = \sum_{\text{all } i,j} a_{ij}\,\alpha_{ij}\ .$$

The ME clearly is equal to the sum of all the vertical and horizontal bond strengths in the matrix where the strength of a bond between two horizontally or vertically adjacent elements

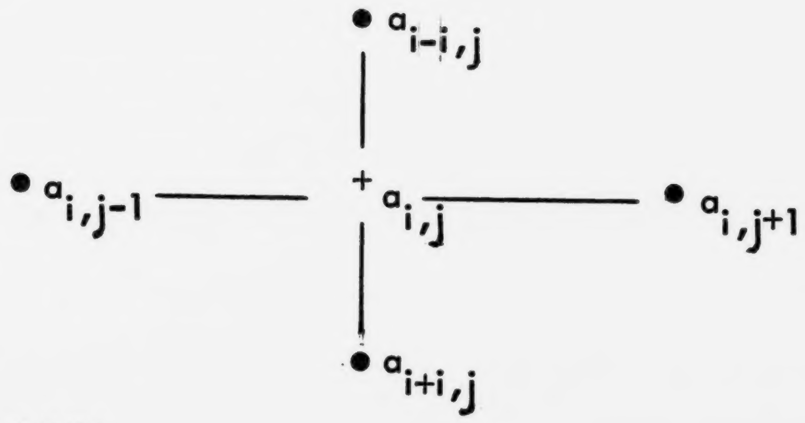
---

[1] The theory and development of this algorithm are due to Dr. W.T. McCormick, Jr.

[2] This algorithm is so called because its measure of effectiveness involves products of nearest neighbor matrix elements that may be likened to bond strengths.

[3] A clumpy matrix is one whose large elements lie near other large elements, forming aggregates called clumps.

[4] With the convention $a_{0,j} = a_{i,0} = a_{i,N+1} = a_{M+1,j} = 0$.



11-12-69-3

FIGURE 1. Representation of Bond Energy ME

is defined as the product of the elements. A slightly more general form of this ME is presented in the more detailed description in Part II of this paper.

To obtain maximum "clumpiness" of the matrix it is necessary to maximize the ME over all row permutations and column permutations of the matrix, i.e.,

$$\max_{\substack{\text{all row perm} \\ \text{\& col perm}}} \left\{ \text{ME} = \sum_{\text{all } i,j} a_{ij}\ \alpha_{ij} \right\}$$

This problem can be formulated equivalently as two quadratic assignment problems[5] and its optimal solution can be determined. However, this rigorous solution is quite time consuming so a suboptimal algorithm has been developed. The suboptimal algorithm is a sequential selection procedure that has proven to be efficient and rapid. The description and details of this technique are contained in Chapter I of Part II.

A simple example will illustrate the sensitivity of the ME and the utility of a rearrangement of the matrix data. Suppose we have a symmetric matrix showing certain associations or relationships between entities A, B, C and D. The initial relationship matrix is shown in Fig. 2a, where the ones in the i,j$^{\text{th}}$ elements of the matrix represent the existence of relationships between entities i and j and the zeros indicate the absence of relationships. It is clear from the definition of the ME and the observation that there are no bonds, that the ME = 0 for the matrix in Fig. 2a. Figures 2b, 2c, 2d, and 2e show progressively greater levels of clumpiness and their MEs are 2, 4, 6, and 8, respectively. Application of the Bond Energy Algorithm produces the ordering shown in Fig. 2e, where it is clear that two clusters have been

---

[5] See Appendix A.

	A	B	C	D
A	1	0	1	0
B	0	1	0	1
C	1	0	1	0
D	0	1	0	1

FIGURE 2a. ME=0

	A	B	C	D
A	1	0	1	0
B	0	1	0	1
D	0	1	0	1
C	1	0	1	0

FIGURE 2b. ME=2

	A	B	C	D
A	1	0	1	0
C	1	0	1	0
B	0	1	0	1
D	0	1	0	1

FIGURE 2c. ME=4

	A	C	B	D
A	1	1	0	0
B	0	0	1	1
D	0	0	1	1
C	1	1	0	0

FIGURE 2d. ME=6

	A	C	B	D
A	1	1	0	0
C	1	1	0	0
B	0	0	1	1
D	0	0	1	1

FIGURE 2e. ME=8

11-12-69-4

FIGURE 2. Illustration of the Sensitivity of the Bond Energy ME

uncovered and in fact the entities have been factored into two unrelated and distinct groups (i.e., A, C and B, D).

This simple example gives an indication of how the Bond Energy Algorithm can produce clearer and deeper understanding of the matrix data by simple rearrangement.

B. THE MOMENT ORDERING ALGORITHM[6]

The purpose of the Moment Ordering Algorithm is to identify the single dominant relationship in an array of data, and to reorder the rows and columns of the array to produce a ranking under this dominant relationship. That is to say, the algorithm finds the principal axis[7] for the data, and arranges both the rows and the columns according to the implicit underlying variable corresponding to the axis. The concept may be made clearer by considering the two examples discussed in Part II. One example involves the distribution of pottery types in a group of archeological sites. The underlying variable is the age of the site, and the algorithm therefore produces a chronological ordering of the sites. The second example involves the voting patterns of a group of Senators. The algorithm determines that the underlying variable is the degree of liberalism/conservatism, and therefore orders the Senators (and the bills voted upon by them) along a liberal/conservative spectrum.

The underlying idea behind the algorithm is the fact that if two rows are very similar to each other their mean row moments should be close to each other in value. The mean row moment x_i of the ith row is defined as

$$x_i = \sum_{j=1}^{N} j a_{ij} \Big/ \sum_{j=1}^{N} a_{ij},$$

where a_{ij} is the ij^{th} element in the array. Similarly, if two columns are closely related, their mean column moments, defined analogously, should be close in value. The algorithm, then, is an attempt to use these moments to rearrange the array so that rows (or columns) are as near as possible to other similar rows (or columns).

The algorithm begins by computing the row moments for the array in its initial state, and placing the rows in ascending order of their moments. The column moments are then calculated, and the columns reordered according to their moments. This reordering, however, changes the values of the row moments. The row moments are therefore recalculated and the rows reordered. The procedure is continued, alternating between row and column reorderings, until an ordering is reached in which both the rows and columns are arranged in order of their moments. Such a stable state is considered a solution. The principal output of the algorithm is then the one-dimensional ordering of the entities on the array axes on the basis of whatever dominant relationship may exist in the data.

---

[6] The initial idea for this algorithm and for this research paper is due to Dr. John J. Martin. The algorithm was improved and developed by Dr. Stephen B. Deutsch.

[7] A principal axis may be thought of as an "underlying variable" by means of which the explicit variables can be listed in a one-dimensional ordering.

As an example of how the Moment Ordering Algorithm operates on a sample data array, consider the relationship matrices given in Fig. 3. When the algorithm is applied to the data array of Fig. 3a, the new array shown in Fig. 3b is obtained. Similar rows are now adjacent to each other, and the overall ordering of the rows reflects their placement along the principal axis of the array. Note the concentration of the non-zero elements along the main diagonal of the new array. This concentration is a property of solutions found by the algorithm. The details of this method and some examples which have been successfully handled are presented in Part II.

	A	B	C	D	E
A	1	1	0	0	1
B	1	1	0	1	0
C	0	0	1	0	1
D	0	1	0	1	0
E	1	0	1	0	1

FIGURE 3a.

	D	B	A	E	C
D	1	1	0	0	0
B	1	1	1	0	0
A	0	1	1	1	0
E	0	0	1	1	1
C	0	0	0	1	1

FIGURE 3b.

11-12-69-5

FIGURE 3. An Example of the Moment Ordering Algorithm

C. THE MOMENT COMPRESSION ALGORITHM[8]

The Moment Compression Algorithm is designed to identify natural groups and clusters of entities by factoring the data relationship matrix into a number of pieces. The algorithm accomplishes this by finding the data ordering which minimizes a specific ME. The ME used by the Moment Compression Algorithm is just the sum of all the row and column second moments about their respective means, that is

$$ME = \sum_{i=1}^{M} r_i + \sum_{j=1}^{N} c_j,$$

where r_i and c_j are the i^{th} row moment and j^{th} column moment. The minimization of this ME over all row and column permutations has the effect of compressing the data in such a way as to force the non-zero matrix elements toward a block-factored form.

---

[8] The theory and development of this algorithm are due to Dr. Paul J. Schweitzer.

This ME was devised because of the observation that the rows and columns of a matrix in perfect block-factored form, when contrasted with the same matrix after row or column permutations, have the smallest sum of the moments of inertia about their means. That is, any row or column permutation of a matrix in perfect block form will "expand" a block and make it less dense, thereby increasing the matrix's total moment of inertia. A matrix in perfect block-factored form is shown in Fig. 4.

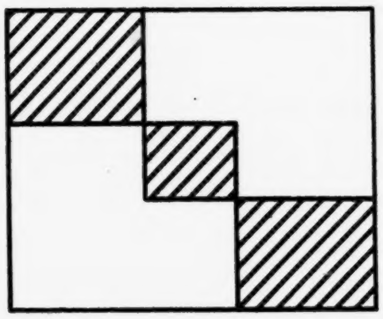


FIGURE 4. Matrix with Perfect Block Form

The problem of ME maximization can be posed as two quadratic assignment problems; however, in practice, it has been solved sub-optimally by an iterative gradient procedure involving linear assignment problems.

When the Moment Compression Algorithm is applied to any of the matrix orderings of Fig. 2 the resulting ordering is the completely block-factored form shown in Fig. 2e. In this special case when the matrix is completely block factorable, the Bond Energy and the Moment Compression Algorithms will both produce block-factored form.

D. CONTRASTS AMONG THE THREE ALGORITHMS

In order to understand better exactly how the three algorithms differ, it is useful to compare their objectives and their computational methods.

The single objective of the Moment Compression Algorithm is to identify groups or clusters by rearrangement of the matrix data. In addition to sharing this objective the Bond Energy Algorithm has the additional objective of determining whether and in what manner these groups are related to one another.[9] Computationally, the MEs of the two algorithms differ substantially in that the Bond Energy ME depends on nearest-neighbor interactions while the Moment Compression ME is completely global. A consequence of this difference is that the Bond Energy ME more adequately describes the topological properties of clumpiness, denseness and connectedness. Another consequence is the greater computational ease in optimizing the Bond Energy ME by use of a rapid sequential selection algorithm which exploits its nearest neighbor dependency.

---

[9] See discussion before Fig. 6.

The Moment Ordering Algorithm differs markedly from both of the previous data ordering methods. Instead of attempting to identify groups, clusters or group interrelationships, the main objective of the Moment Ordering Algorithm is to produce a one-dimensional ordering of entities along the axes of the matrix. It accomplished this by finding a dominant variable or principal axis along which these entities can be ordered. Computationally, like the Moment Compression Algorithm, the Moment Ordering Algorithm employs moments which are global matrix measures, and thus it is not as sensitive to local details as the Bond Energy Algorithm. Its principal computational difference, though, from the Bond Energy and the Moment Compression Algorithms is that it is a completely heuristic iterative technique that does not attempt to optimize any measure of effectiveness.

III. CONCLUSIONS

The following statements are the general assessments and conclusions regarding the applicability, overall usefulness, and efficiency of the three algorithms developed for direct analysis of multivariate systems by matrix reordering.

- The Bond Energy Algorithm proved to be the most generally useful and versatile of the three algorithms for treating certain problems of multivariate analysis. It is capable not only of classifying and clustering data but also of successfully identifying areas of interrelationships that exist among these clusters. It has been found to be an efficient and general approach to problems involving clusters and group structures.

- The Moment Ordering Algorithm is an efficient technique for uncovering and displaying a univariate relationship inherent in the data. That is, it is a fast and direct method for uncovering the principal axis of a data structure. The efficiency of the algorithm was found to be in direct proportion to its ultimate success in identifying a principal axis. The primary utility of this algorithm is in determining a good one-dimensional ordering of the data rather than in uncovering clusters or group interrelationships in the data.

- The Moment Compression Algorithm is successful at identifying clusters and groups inherent in the data. Both it and the Bond Energy Algorithm will put a matrix into block form, if this is possible. However, the Moment Compression Algorithm is slower and therefore less useful for large problems. Unlike the Bond Energy Algorithm, the Moment Compression Algorithm cannot handle the case of block-checkerboard[1] matrices arising from multilateral group relationships. Consequently the Moment Compression Algorithm is considered inferior to the Bond Energy Algorithm both with regard to computational speed and versatility of its measure of effectiveness.

---

[1] See Fig. 6.

PART II: DETAILED DESCRIPTIONS AND APPLICATIONS

I. THE BOND ENERGY ALGORITHM

A. MOTIVATION

The motivation for the development of the Bond Energy Algorithm was to be able to treat a broader class of problems than that normally found in cluster analysis applications. In addition, it was desired to operate directly on and manipulate the original data without creating or losing information. The object is not only to classify and group similar entities but also to determine how and by what means these groups are interrelated. This can be illustrated by considering a symmetric binary (0-1) relationship matrix between N entities. If the N entities can be separated into, say, four unique groups (unique meaning that the entities in one group are related only among themselves and not with any entities outside their own group), then many of the techniques of cluster analysis are applicable. In this case it is possible to reorder the rows and columns of the input data matrix to obtain the form given in Fig. 5.

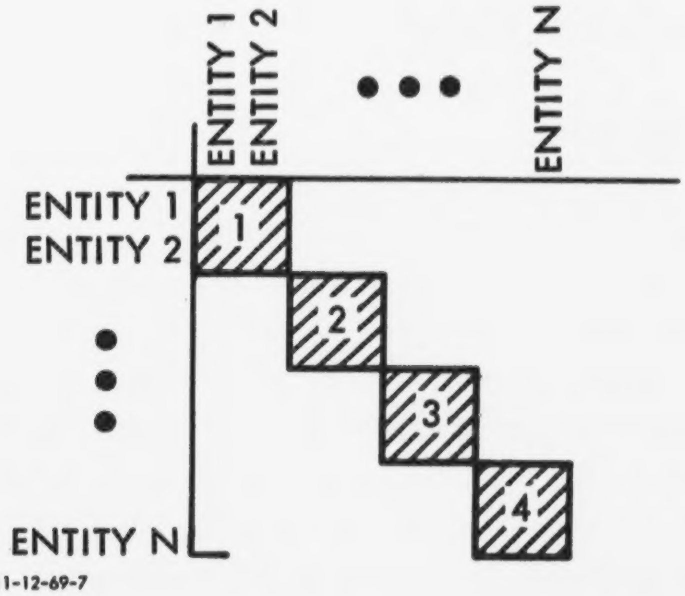


FIGURE 5. Relationship Matrix Showing 4 Unique Groups

However, if the entities are not completely factorable into unique groups then it is often desirable to identify not only the principal groups but also their significant areas of relationship. In other words, it might be desirable to rearrange the data matrix to obtain a checkerboard pattern if it is possible. This type of pattern is shown in Fig. 6, where the off-diagonal blocks of large Xs represent data clumps containing a sizable percentage of non-zero entries, thus indicating partial or total intergroup relationships.

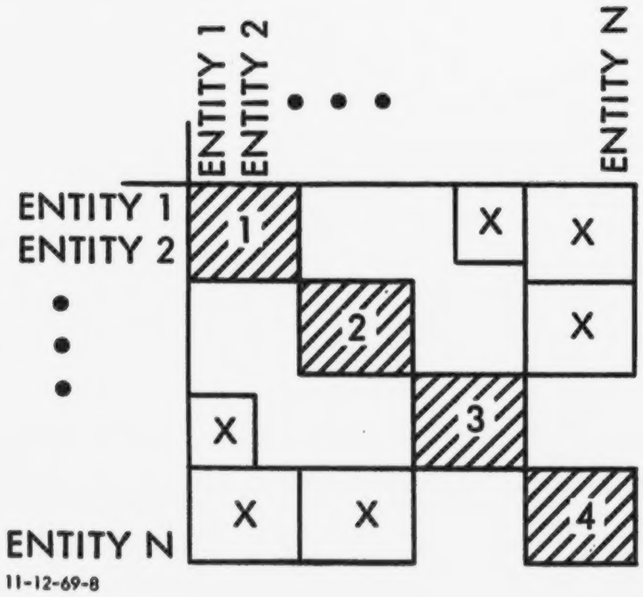


FIGURE 6. Relationship Matrix with Block-Checkerboard Form

The essential question is, given a matrix where the data are presented in an arbitrary manner, how can the rows and columns of a matrix be simply rearranged to obtain as "clumpy" a matrix form as possible.

B. THE MEASURE OF EFFECTIVENESS

1. Definition and Interpretations

In order to analytically determine the "clumpiness" of a particular matrix, it was necessary to develop some measure of effectiveness (ME) [1] of how any subsequently proposed algorithm would progress. This ME must be sensitive to and depend on local clumpiness while also characterizing the clumpiness of the entire matrix. The essential idea behind the ME, which fulfills this requirement, came from likening the situation to that of the saturation of bonds in the nucleus of an atom. That is, when the nucleons are clumped together there is total bond saturation in the interior of the nucleus while the bonds of the nucleons near the surface are unsaturated. The intent was to find an ME which when maximized, resulted in as few unattached or unbonded matrix elements as possible. The bond strength between two adjacent matrix elements is defined as the $1/k^{th}$ power of the product of the matrix elements. Maximization of the ME will maximize the sum of all the bond strengths, and therefore clump together the larger non-zero matrix elements. Another physical phenomenon that may be likened to this situation is that of water beads on a glass. The beads tend to aggregate into larger clumps in order to minimize the surface energy. The ME can be defined, then, as just the sum of all the bond strengths in a matrix. Thus

---

[1] A more complete discussion of MEs may be found in Appendix G.

$$\mathrm{ME}_k = \sum_{\text{all } i,j} a_{ij}^{1/k'} \, {}_k\alpha_{ij}$$

where[2]

$${}_k\alpha_{ij} = \frac{1}{2}\left[a_{i,j+1}^{1/k} + a_{i,j-1}^{1/k} + a_{i+1,j}^{1/k} + a_{i-1,j}^{1/k}\right]$$

and k is a weighting constant, which is usually set equal to 2. The ME may be interpreted mathematically as the sum of the scalar products (or projections on one another) of all the contiguous row vectors[3] plus the sum of the scalar products of all of the contiguous column vectors.[3]

2. Normalization of the ME

The ME defined above can be normalized so that its value varies between 0 and 1. This normalized measure of effectiveness (NME) is defined as

$$\mathrm{NME}_k = \frac{1}{S} \sum_{\text{all } i,j} a_{ij}^{1/k} \, {}_k\alpha_{ij}$$

$$\text{and} \qquad 0 \leqslant \mathrm{NME}_k < 1$$

where

S is the normalization constant defined as

$$S = 2 \sum_{\text{all } i,j} a_{ij}^{2/k}.$$

S can be interpreted mathematically as the sum of the squares of the L_2 norms[4] of all the row and column vectors. The advantage in having a normalized ME is that it is easier to determine how much improvement in the clumpiness of a matrix has been achieved since it is a measure

---

[2] Again, $a_{0,j} = a_{i,0} = a_{M+1,j} = a_{i,N+1} = 0$.

[3] The i^{th} row vector is comprised of the elements $a_{i1}^{1/k}, a_{i2}^{1/k}, \ldots, a_{iN_i}^{1/k}$ in the i^{th} row of the matrix. A column vector is defined analogously.

[4] The L_2 vector norm is defined as

$$\|v\|_2 = \sqrt{\sum_{i=1}^{M} v_i^2}$$

where M is the dimension of the vector space. The fact that the NME_k is properly normalized follows from a basic inequality for a normed space $|\vec{a}|^2 + |\vec{b}|^2 \geqslant 2(\vec{a}, \vec{b})$.

of the amount of bond saturation. For instance, if the NME of the reordered data matrix equals 0.6, whereas the NME of the initial data matrix equals 0.2, then it can be concluded that there does exist a good deal of inherent group structure and interrelationship that was not initially evident. Moreover, the final NME gives an absolute measure of the existence of the clusters that we have sought to uncover.

3. Advantages of the ME

The ME proposed above has some very important theoretical and computational advantages which will be enumerated here.

- The NME can be used for matrices of any size or shape. In addition, symmetry of the matrix is not required. The only restriction is that the matrix elements be non-negative, real numbers.

- Since the vertical (horizontal) bonds are unaffected by interchange of the columns (rows), the ME decomposes into two parts; one (sum of the vertical bonds) dependent only on row permutations, and the other (sum of the horizontal bonds) dependent only on column permutations. Consequently optimization of the ME can be achieved in exactly two passes, one finding the optimal column permutation, the other finding the optimal row permutation.

- These two passes can be carried out completely independently of each other. In particular, it is not necessary to alternate between row and column permutations, as in the Moment Ordering Algorithm, thus eliminating the possibility of any cycling[5] of the solution.

- Since the contribution to the ME from any column (or row) is only affected by the two adjacent columns (or rows), the optimization lends itself very well to a multistage sequential selection process.

- The Bond Energy ME optimization does not require any prior prejudices, such as forcing the data into clumps along the diagonal or forcing the data into block-diagonal form. The representation of the data that is sought is a tight clumped form and so the maximization of the ME might very well allow the possibility of far outlying elements in order to achieve globally higher degree of compactness. This feature is particularly important in the case of multilateral relations between groups of entities where it is clearly not possible to obtain a block-diagonal form.

---

[5] This phenomenon occurs when the solution gets alternately better then worse.

- The 1/k power[6] of a_{ij} appearing in the expression for the ME allows any desired weighting of the larger matrix elements.

C. THE SOLUTION: MAXIMIZATION OF THE ME

1. The Exact Solution

The problem to be solved as implied earlier is to maximize the ME over all row and column permutations. That is,

$$\underset{\pi,\phi}{\text{Max}} \left\{ \frac{1}{2} \sum_{i,j} a^{1/k}_{\pi(i),\phi(j)} \left[a^{1/k}_{\pi(i),\,\phi(j+1)} + a^{1/k}_{\pi(i),\,\phi(j-1)} + a^{1/k}_{\pi(i+1),\,\phi(j)} + a^{1/k}_{\pi(i-1),\,\phi(j)} \right] \right\}$$

where $\pi = \{\pi(1), \pi(2), \ldots, \pi(M)\}$ and $\phi = \{\phi(1), \phi(2), \ldots, \phi(N)\}$

are the row and column permutations. This can be thought of physically as maximizing the sum of all the bond energies and mathematically as maximizing the sum of all the scalar products of contiguous row vectors and column vectors. This maximization problem can be stated equivalently as two quadratic assignment problems (the reader is referred to Appendix A for the detailed formalism). The first seeks a permutation of the columns of $[a_{ij}]$ which maximizes the row bond energy, the other seeks a permutation of the rows of $[a_{ij}]$ which maximizes the column bond energy. These optimizations may be viewed as two clustering procedures, one which reorders the rows on the basis of their similarity (similarity being measured by the scalar product of the two rows), the other reordering the columns. Reassembling the matrix after both clusterings produces the dense blocks shown in Fig. 6. Although quadratic assignment problems can be solved exactly as well as approximately (for exact and approximate methods see the references listed in Appendix A), the solution of this problem requires a large amount of computer time in either case. Our own approximate sequential selection algorithm has been developed which takes advantage of the nearest neighbor properties of the measure of effectiveness.

2. Approximate Solution

a. Description of the Sequential Selection Algorithm. The suboptimal algorithm which has been actually used to determine local optima of the ME is as follows:

---

[6] The sensitivity of the ME to the value of k is discussed in Appendix G.

(1) Compute and store the scalar products of each row with every other row and each column with every other column.

(2) Select any column to begin the selection process. Set i=1.

(3) Next, try each of the remaining N-1 columns placed alongside the first column and compare its contribution[7] to the horizontal bond ME.

(4) Place alongside the first column that particular column which gives the largest contribution to the ME.

(5) Continue the process at the ith step by comparing the contribution to the ME by placing each of the N-i remaining columns in each of the i+1 possible positions,[8] and putting the one which gives the largest contribution to the ME in its proper place.

(6) After the process is completed by placing the last remaining column in its "best" place, then the entire procedure (items 2 through 5) is repeated on the rows. It is, however, not necessary to repeat the procedure on the rows if the initial input matrix is symmetric since the final resulting row order will be identical with the column ordering, yielding a symmetric matrix.

b. Advantages of the Algorithm. The algorithm described above has several attractive advantages which are noted here.

(1) Since the algorithm is finite and non-iterative, there are no convergence problems.

(2) The algorithm will always reduce a matrix to pure block form if it is possible to obtain this form by row and column permutations (see Appendix B for proof).

(3) The solution obtained from the algorithm is independent of the input order of the rows (or columns) but is only dependent on the initial row (or column) chosen to start the sequential selection process.

(4) The results of the algorithm are very insensitive to the starting point (i.e., starting row or column), hence any solution is a "good" one (see Table 2).

(5) The computation time for the algorithm depends only on the size of the matrix and not on its elements.

(6) The algorithm uses no thresholds or filtering during its operation which can alter its course and affect the final result.

(7) Only the raw input data matrix is used to determine the new row and column orderings.

---

[7] The contribution is just the dot product of the chosen column vector with the first column vector.

[8] The i+1 positions are to the left and right of the i columns already placed.

3. An Example

A simple example taken from *Principles of Numerical Taxonomy* (Ref. 15) will illustrate how the algorithm can identify the clusters and their interrelationships. The similarity matrix of Fig. 7 is given where a numerical value of 5 in element i,j indicates a high degree of similarity between entity i and entity j, and 0 indicates no similarity.

	A	B	C	D	E	F	G	H	I	J
A	5	4	1	0	4	1	1	0	3	1
B	4	5	0	1	3	1	1	0	4	1
C	1	0	5	0	1	3	3	0	1	2
D	0	1	0	5	0	0	0	4	0	1
E	4	3	1	0	5	1	0	0	4	1
F	1	1	3	0	1	5	3	0	1	3
G	1	1	3	0	0	3	5	0	1	2
H	0	0	0	4	0	0	0	5	1	0
I	3	4	1	0	4	1	1	1	5	1
J	1	1	2	1	1	3	2	0	1	5

11-12-69-9

FIGURE 7. Initial Non-Binary Similarity Matrix

By applying the algorithm described above, a new axis ordering and a new matrix are obtained and are shown in Fig. 8.

	H	D	B	A	E	I	J	F	G	C
H	5	4	0	0	0	1	0	0	0	0
D	4	5	1	0	0	0	1	0	0	0
B	0	1	5	4	3	4	1	1	1	0
A	0	0	4	5	4	3	1	1	1	1
E	0	0	3	4	5	4	1	1	0	1
I	1	0	4	3	4	5	1	1	1	1
J	0	1	1	1	1	1	5	3	2	2
F	0	0	1	1	1	1	3	5	3	3
G	0	0	1	1	0	1	2	3	5	3
C	0	0	0	1	1	1	2	3	3	5

11-12-69-10

FIGURE 8. Reordered Non-Binary Similarity Matrix

It is easy to identify three major diagonal blocks of large numbers representing three clusters or groups of entities. H and D constitute the first group, B, A, E and I the second group, and J, F, G and C the third group. From the grouping of the smaller off-diagonal elements it is evident that there is some weak relationship between the second and third groups but essentially no relationship between the first group and either of the other two. It is also quite apparent from this example that this new form for the matrix data conveys more information concerning the group structure and relationships than does the original matrix form.

D. OPERATION OF THE ALGORITHM

1. Computing Time Requirements

If the original data matrix is of dimension M by N, then the total number of arithmetic operations necessary to perform all the initial row and column dot products is just:

$$\text{Operations} = N \sum_{i=1}^{M-1} i + M \sum_{j=1}^{N-1} j \quad \text{or,}$$

$$\text{Operations} = N \cdot \frac{M(M-1)}{2} + M \cdot \frac{N(N-1)}{2} \quad \text{or}$$

for large M and N,

$$\text{Operations} \approx \frac{M^2N + N^2M}{2}.$$

At step i of the algorithm, it is necessary to compare the contribution of the ME of all the remaining N-i unplaced columns in the i+1 possible positions, thus the total number of column comparisons equals

$$\sum_{i=1}^{N-1} (i+1)(N-i) = \sum_{i=1}^{N-1} iN - i^2 + N - i$$

$$\approx \frac{N^3}{6} \quad \text{for large N.}$$

Similarly it requires approximately $M^3/6$ comparisons for the rows. Thus for a square matrix, the computation time of the algorithm goes as N^3. This theoretical variation in the computing time has been borne out experimentally as can be seen in Table 1. The computing time[9] in seconds is given for various size problems (times given are for a single starting point).

---

[9] On CDC 1604 computer using University of Minnesota compiler.

Table 1. BOND ENERGY ALGORITHM COMPUTATION TIME

N	M	Time
21	21	11 sec
29	29	23 sec
48	48	124 sec

From this data a scaling law can be derived which gives the required computation time in seconds for a given size square matrix, for a single starting point.

$$\text{Computation Time (sec)} = 0.0012\ N^3$$

2. Ties

It occasionally happens that ties occur during the course of the sequential selection algorithm. Ties between rows and columns can occur in the following ways:

(1) Tie arising from putting the same as yet unplaced column (or row) in two or more possible positions.
(2) Tie arising from putting different as yet unplaced columns (or rows) in the same positions.
(3) Ties arising from putting different as yet unplaced columns (or rows) in different possible positions.

We have no present criterion for deciding how to break ties arising from condition (1), nor is it known whether there is reason to select one alternative over the others. Ties arising from conditions (2) and (3) are broken by selecting the unplaced row or column which has the shortest length in the L_2 norm.[10] Thinking in terms of the ME mathematically, if we can obtain the same scalar products or projections with two vectors, then the shorter should be used rather than the larger one. This tie-breaking mechanism has been found to work satisfactorily in that it leads to informative final data arrangements.

3. Effect of Starting Point

Although the results of the algorithm do not depend on the order in which the rows and columns are considered, there is a difference in the final results depending on which row or column is selected to start the multistage decision process. In the example presented in Figs. 7 and 8, the problem was started 10 times, beginning once with each column. Table 2 gives the frequency of occurrence and final ME for each distinct solution.

---

[10] This is just the square root of the sum of the squares of all the elements of the vector.

Table 2. FREQUENCY DISTRIBUTION OF ME

Solution No.	Frequency	ME
1	3	419
2	3	419
3	1	414
4	2	414
5	1	412

Several significant facts may be noted from these results. First of all, the solutions with the highest ME, 419, which are believed to be the globally optimum solutions, occur 60 percent of the time. The difference between the best and worst solution is only 7 out of over 400, or less than 2 percent. A noteworthy point here is that the final "solution" (ME) depends very weakly on the starting point and even the worst "solution" is not very far from the optimal solution. With regard to the final group structure, it has been found that the various near optimal solutions do not differ significantly from the optimal solution. The various solutions are due to the rearrangement of the entities within a cluster group and the reordering of the groups themselves. These results have been confirmed by experimentation on significantly larger matrices.

4. Formatting Data

The input format for the data can be in any matrix form. This means that the Bond Energy Algorithm permits analysis of the raw data without forming a similarity matrix.[11] For example, suppose we have an object-attribute matrix and we desire to find out which objects are similar. The advantage of performing the grouping directly upon the object-attribute matrix, rather than upon the similarity matrix, is that it is now possible to determine which attributes characterize a particular group of objects (see example 4).

E. APPLICATIONS

Several applications of this method have already been attempted and others have been suggested. It appears that the algorithm is applicable to a wide class of problems, a number of which will be enumerated here.

(1) Identification of natural groups and subgroups within data.
(2) Identification of relationships and dependencies between groups.
(3) Relationships of groups of attributes to groups of objects.
(4) Examining influence relationships and structures via nonsymmetrical data matrices.

---

[11] A similarity matrix is a symmetric matrix whose i,j^{th} element is a measure of the similarity of entity i to entity j. Applying the Bond Energy Algorithm to a similarity matrix identifies (as the diagonal blocks) the main groupings of entities and (as the off-diagonal clumps) the intergroup relationships.

(5) Analysis of hierarchical clustering and grouping via quantified numerical relationships.
(6) Factoring of large linear assignment problems (Ref. 32).
(7) Factoring of large management problems to identify optimal subtasks.
(8) Clustering of correlation matrices.
(9) Solution of traveling salesman problems (Ref. 31).
(10) Unscrambling flow graphs and network relationships.

F. EXAMPLES

A number of examples are presented in the following paragraphs to illustrate the operation and the potential application of the Bond Energy Algorithm. It should be clearly understood that the algorithm operates on matrices that contain "hard" numerical entries and therefore considers each data matrix to be an exact representation of the relationships involved. We feel, nevertheless, that the algorithm has application for problems involving "soft" data (Airport example) as well as "hard" data (Hindi consonant example), as long as proper care is taken to judiciously weigh the results subject to the degree of validity of the input information.

1. Example 1

Bonner (Ref. 3) has presented several clustering techniques which uncover group structure in matrix data. For this example, the Bond Energy Algorithm is applied in several different ways to illustrate its advantages and directness for gathering similar data into clusters. The objects which will be clustered are defined by a set of attributes which characterize them.

Bonner presents a binary description of an object set as an object-attribute matrix which is shown in Fig. 9.

		ATTRIBUTE NUMBER					
		1	2	3	4	5	6
OBJECT NUMBER	1	1	0	0	1	0	0
	2	1	1	0	1	0	0
	3	0	0	1	1	1	1
	4	0	1	1	0	0	1
	5	1	0	0	1	1	0
	6	0	0	1	0	1	0
	7	0	1	0	1	0	1
	8	1	1	1	0	0	0

11-12-69-11

FIGURE 9. Initial Binary Object-Attribute Matrix

He then proceeds to form a similarity matrix P, where the P_{ij} are defined as

$$P_{ij} = \frac{C_{ij}}{C_{ii} + C_{jj} - C_{ij}}$$

and C_{ij} is the number of attributes which are "one" for both object i and object j. The similarity matrix corresponding to Fig. 9 is shown in Fig. 10. A threshold T=0.45 is then used to convert the fractional similarity matrix of Fig. 10 to a binary similarity matrix by setting those matrix elements to one whose values are greater than 0.45 and the rest equal to zero. This similarity matrix is shown in Fig. 11. Bonner then uses this similarity matrix as a starting point for several clustering techniques.

		OBJECT NUMBER							
		1	2	3	4	5	6	7	8
	1	1	2/3	1/5	0	2/3	0	1/4	1/4
	2		1	1/6	1/5	2/4	0	2/4	2/4
	3			1	2/5	2/5	2/4	2/5	1/6
OBJECT NUMBER	4				1	0	1/4	2/4	2/4
	5					1	1/4	1/6	1/5
	6						1	0	1/4
	7							1	1/5
	8								1

11-12-69-12

FIGURE 10. Initial Fractional Object Similarity Matrix

		OBJECT NUMBER							
		1	2	3	4	5	6	7	8
	1	1	1	0	0	1	0	0	0
	2	1	1	0	0	1	0	1	1
	3	0	0	1	0	0	1	0	0
OBJECT NUMBER	4	0	0	0	1	0	0	1	1
	5	1	1	0	0	1	0	0	0
	6	0	0	1	0	0	1	0	0
	7	0	1	0	1	0	0	1	0
	8	0	1	0	1	0	0	0	1

11-12-69-13

FIGURE 11. Initial Binary Similarity Matrix

The Bond Energy Algorithm has several advantages over Bonner's technique. First, it is able to operate directly on the object-attribute matrix without forming a similarity matrix thus permitting it to identify those particular attributes that characterize objects in the same cluster. Second, the application of the algorithm will never result in a loss of information since the data are only rearranged. Finally, even using a similarity matrix the algorithm can produce a reordering which not only displays the clusters but also their strengths and relationships. These advantages will be demonstrated by successive application of the Bond Energy Algorithm to the matrices of Figs. 9, 10 and 11.

When the object-attribute matrix of Fig. 9 is rearranged by the Bond Energy Algorithm, the new data matrix of Fig. 12 is obtained. When rectangles are constructed around solid blocks of 1s in two or more rows and columns, it can be seen that the objects fall into 4 "core" clusters: 3,6 and 2,1,5, and 4,7, and 8. It is also observed that attributes 3 and 5 are the essential characterizing attributes of the 3,6 object cluster, attributes 4 and 1 are the characterizing ones for the cluster containing objects 2,1 and 5, and attributes 2 and 6 characterize the cluster containing objects 4 and 7.

		ATTRIBUTE NUMBER					
		4	1	2	6	3	5
OBJECT NUMBER	8	0	1	1	0	1	0
	4	0	0	1	1	1	0
	7	1	0	1	1	0	0
	2	1	1	1	0	0	0
	1	1	1	0	0	0	0
	5	1	1	0	0	0	1
	3	1	0	0	1	1	1
	6	0	0	0	0	1	1

11-12-69-14

FIGURE 12. Reordered Binary Object-Attribute Matrix

When the Bond Energy Algorithm is applied to the fractional object similarity matrix of Fig. 10, a new ordering is obtained. In this new ordering in Fig. 13,

		OBJECT NUMBER							
		6	3	5	1	2	7	4	8
OBJECT NUMBER	6	1	2/4						
	3	2/4	1						
	5			1	2/3	2/4			
	1			2/3	1	2/3			
	2			2/4	2/3	1	2/4		2/4
	7					2/4	1	2/4	
	4						2/4	1	2/4
	8					2/4		2/4	1

FIGURE 13. Reordered Fractional Object Similarity Matrix

only the larger elements (i.e., 1/2 or greater) are shown so that the clusters can be more easily identified. Again, it is possible to identify the clusters and how they interrelate. Objects 3 and 6 form a very tight independent cluster. Objects 5,1,2 form another tight cluster, although there is a non-trivial relationship between object 2 and objects 7 and 8. Objects 4 and 7 form another cluster that is somewhat related to objects 2 and 8. Thus, visually, this form of data presentation is helpful and its computational requirements are very small (less than one second).

When the Bond Energy Algorithm is applied to the binary similarity matrix of Fig. 11, the result is given in Fig. 14. It is quite apparent that these results illustrate the same relationships and clusters as those shown in Fig. 13, but are inferior since the strengths of the relationships are not shown. This illustrates that while Bonner's filtering technique leads to the uncovering of major clusters, it also loses information present in the original data matrix.

OBJECT NUMBER

OBJECT NUMBER	3	6	1	5	2	7	4	8
3	1	1	0	0	0	0	0	0
6	1	1	0	0	0	0	0	0
1	0	0	1	1	1	0	0	0
5	0	0	1	1	1	0	0	0
2	0	0	1	1	1	1	0	1
7	0	0	0	0	1	1	1	0
4	0	0	0	0	0	1	1	1
8	0	0	0	0	1	0	1	1

11-12-69-17

FIGURE 14. Reordered Binary Similarity Matrix

2. Marketing Techniques and Applications

In displaying the data relationships in this example, it is found that the application of the Bond Energy Algorithm reveals several latent group associations and significantly enhances the quality of the presentation of the data.

Figure 15 contains a matrix showing which Marketing Techniques are used for particular Marketing Applications.[12] By application of the algorithm it is possible to reorder or relist the marketing applications on the one axis and the marketing techniques on the other

---

[12] The data for this example were taken from the September-October 1969 issue of the Harvard Business Review from "Techniques in Marketing Research" by J.F. Dash and C. Berenson.

	ADVERTISING RESEARCH	ACQUISITION SCREENING	BRAND STRATEGY	CUSTOMER SEGMENTATION	CUSTOMER SERVICE	DISTRIBUTION PLANNING	MARKET SEGMENTATION	PRICING STRATEGY	PRODUCT LIFE-CYCLE ANALYSIS	PRODUC LINE ANALYS
REGRESSION & CORRELATION ANALYSIS	X		X				X	X		
DISCOUNTED CASH FLOW (DCF)						X				X
INCREMENTAL ANALYSIS	X					X		X		
MULTIPLE REGRESSION/ CORRELATION	X		X							
RANDOM SAMPLING										
SAMPLING THEORY	X		X							
BAYESIAN APPROACH	X							X	X	
COST-BENEFIT ANALYSIS	X									
CRITICAL PATH METHOD (CPM)										X
DECISION TREES	X	X						X		X
DYNAMIC PROGRAMMING	X									
EXPONENTIAL SMOOTHING										
INDUSTRIAL DYNAMICS						X		X		
INPUT-OUTPUT ANALYSIS										X
LINEAR PROGRAMMING	X					X				
MARKOV PROCESS			X	X						
MONTE CARLO SIMULATION		X	X			X				X
NONLINEAR PROGRAMMING	X					X				
NUMERICAL TAXONOMY	X		X	X			X			
PERT										X
QUEUEING MODELS					X	X				
RISK ANALYSIS	X	X						X		X
SENSITIVITY ANALYSIS	X	X						X		X
TECHNOLOGICAL FORECASTING							X		X	X

11-12-69-25

FIGURE 15. In

OMER ICE	DISTRIBUTION PLANNING	MARKET SEGMENTATION	PRICING STRATEGY	PRODUCT LIFE-CYCLE ANALYSIS	PRODUCT LINE ANALYSIS	PRODUCT PLANNING	R&D PLANNING	ROI ANALYSIS	SALES FORECASTING	TEST MARKETING	VENTURE PLANNING
		X	X						X		
	X				X		X	X			
	X		X				X				
									X		
									X	X	
			X	X		X		X			
								X			
					X	X	X			X	X
			X		X						
							X	X			X
									X		
	X		X								
					X	X	X		X		X
	X										
	X				X						X
	X										
		X									
					X	X	X			X	X
X	X										
			X		X						
			X		X						
		X		X	X	X	X		X		X

FIGURE 15. Initial Matrix of Marketing Techniques and Applications

	CUSTOMER SEGMENTATION	MARKET SEGMENTATION	SALES FORECASTING	BRAND STRATEGY	ADVERTISING RESEARCH	PRICING STRATEGY	ACQUISITION SCREENING	PRODUCT LINE ANALYSIS	VENTURE PLANNING
MARKOV PROCESSES	X			X					
NUMERICAL TAXONOMY	X	X		X	X				
REGRESSION & CORRELATION ANALYSIS		X	☒	☒	☒	X			
MULTIPLE REGRESSION/ CORRELATION			☒	☒	☒				
SAMPLING THEORY			☒	☒	☒				
EXPONENTIAL SMOOTHING			X						
INPUT-OUTPUT ANALYSIS			X					☒	☒
TECHNOLOGICAL FORECASTING		X	X					☒	☒
CRITICAL PATH METHOD (CPM)								☒	☒
PERT								☒	☒
MONTE CARLO SIMULATION				X			X	X	X
DISCOUNTED CASH FLOW (DCF)								X	
DYNAMIC PROGRAMMING					X				X
COST-BENEFIT ANALYSIS					X				
BAYESIAN APPROACH					X	X			
DECISION TREES					☒	☒	☒	☒	
RISK ANALYSIS					☒	☒	☒	☒	
SENSITIVITY ANALYSIS					☒	☒	☒	☒	
INCREMENTAL ANALYSIS					X	X			
INDUSTRIAL DYNAMICS						X			
QUEUEING MODELS									
LINEAR PROGRAMMING					X				
NONLINEAR PROGRAMMING					X				
RANDOM SAMPLING									

11-12-69-26

FIG

RICING TRATEGY	ACQUISITION SCREENING	PRODUCT LINE ANALYSIS	VENTURE PLANNING	R&D PLANNING	PRODUCT PLANNING	PRODUCT LIFE-CYCLE ANALYSIS	ROI ANALYSIS	DISTRIBUTION PLANNING	CUSTOMER SERVICE	TEST MARKETING
X										
										X
		☒	☒	☒	☒					
		☒	☒	☒	☒	X				
		☒	☒	☒	☒					X
		☒	☒	☒	☒					X
	X	X	X					X		
		X		X			X	X		
			X	X			X			
							X			
X					X	X	X			
☒	☒	☒								
☒	☒	☒								
☒	☒	☒								
X				X				X		
X								X		
								X	X	
								X		
								X		

FIGURE 16. Reordered Matrix of Marketing Techniques and Applications

axis, while preserving all the data relationships contained in the original matrix. Fig. 16 contains the reordered matrix produced by the Bond Energy Algorithm. With the data in this new matrix form, it is now possible to identify three major clusters or clumps of data. It is believed that in this new form it is possible to uncover useful information that was not obvious from the original matrix.

First, the algorithm groups marketing analysis techniques that are used for the same applications and also it groups marketing applications that utilize the same marketing techniques. This has the effect of putting similar marketing techniques near one another on the vertical axis and putting similar applications together on the horizontal axis. It is postulated that the clumps provide, for one thing, a basis for efficient assignment of responsibilities to analysts and their supervisors, and for another, by exception, a basis for deciding upon the relative merits of "techniques" specialists and "applications" specialists.

Second, if it is possible to factor a matrix completely so that it is apparent that there is a unique relationship between a certain group of marketing techniques and a certain group of marketing applications, then the algorithm will accomplish this. In this example, this has been partially done by identifying three more or less independent clumps in Fig. 16. In particular, as was noted by the authors, PERT and CPM are similar in concept and hence they occur together in the same clump. Also, as noted in the article, risk analysis is often used in conjunction with the method of decision trees. Here again, these marketing techniques are contiguous in the new ordering. On the other axis it is found that similar "marketing applications" are grouped together. For example, Product planning, R&D planning, Venture planning and Product-line analysis all involve planning of some sort and occur in the same clump because they utilize common "marketing techniques" for planning, such as PERT, CPM, etc.

Another possible way in which the clumped matrix of Fig. 16 can be useful is to suggest possible unexploited application of techniques to marketing applications to which they have not already been applied. These could be identified by looking for conspicuous holes within the clumps or omissions on the borders of the clumps.

Thus, it appears in this example, that with proper arrangement of binary (yes-no) or quantified data given in matrix representation, that the amount of information conveyed can be significantly enhanced to such an extent that it is undesirable to present it in other than clustered form.

3. Coordinating Airport Design[13]

A practical way to design an airport is to factor the problem into a number of smaller pieces. If the subproblems can be solved separately and then adjusted so as to remain valid in the context of the original problem, then the task is completed. It is, however, necessary to determine the best way to factor the big problem into more manageable pieces.

A numerical example will illustrate the applicability of the Bond Energy Algorithm to the problem. The objective is to exploit the structure of an airport problem in such a way as to identify two things:

- The "natural" subproblems
- The necessary coordination between subproblems.

The ultimate accomplishment would be to factor the problem into small, completely independent subproblems. But given that complete independence is impossible, the next best thing is to minimize the intergroup dependencies by identifying the optimal way to subdivide the problem.

The first step is to describe the airport problem in terms of a set of variables and their interrelations. A partial list of exogenous and control variables is shown in Table 3.

The exogenous variables describe those factors mostly dictated by the environment while the control variables apply to those factors primarily under control of an airport planner. Let X_i be the i^{th} exogenous variable and let D_i be the i^{th} control variable. The X_i's may be thought of as input data and D_i's as the design decisions. Given a set of values for the X_i's, it is assumed that there exists some way of measuring the performance of an airport design based on some criteria. The details of the performance function are not needed; just a few basic characteristics. Let P be the measure of performance and let F be the function that measures performance. Clearly, P is a function of the D_i's, hence

$$P=F(D_1, D_2 \ldots, D_{27}).$$

F will, in general, depend on the X_i's; however, the discussion will be limited to a specific set of values for the X_i's. The design problem involves selecting values for the D_i's that maximize

---

[13] The analysis and the data for this application are due to Mr. T.W. White of the Institute for Defense Analyses.

Table 3. AIRPORT VARIABLES

Exogenous Variables

1. Total air travel demand
2. Originating passengers
3. Transferring passengers
4. Terminating passengers
5. Greeters and well-wishers
6. Access ground transportation mode for passengers
7. Egress ground transportation mode for passengers
8. Airport employees
9. Taxis and cars that do not park
10. Cars whose drivers park and fly
11. Rental cars going to the airport
12. Rental cars driven from the airport
13. Bus and limousine
14. Employee access transportation mode
15. Passenger trip duration
16. Aircraft turn around time on apron
17. Mix of aircraft by capacity
18. Gate schedule: aircraft arrivals and departures
19. Origin/destination pattern for baggage at airport
20. Air cargo demand
21. Runway demand

Control Variables

1. Passenger check-in
2. Baggage check-in
3. Baggage claim
4. Baggage moving system
5. Intra-airport transportation system
6. Cargo terminal
7. Close-in parking lots
8. Remote parking lots
9. Main access roads to and from airport
10. Circulation roads within airport
11. Service area for rental cars
12. Parking lots for rental cars
13. Curb space for unloading
14. Curb space for loading
15. Waiting areas at gates
16. Stations for intra-airport transportation system
17. Aircraft loading system
18. Concessions
19. Rental car desk
20. Runway capacity
21. Number of gates
22. Passenger information
23. Cargo transfer
24. Air traffic control system
25. Refuse removal
26. Flight operations and crew facilities
27. Aircraft service on the apron

P. The problem can be simplified, for example, if the function F "factors" into two parts; that is, if there are two functions F_a and F_b and if the D_i's can be split into two groups such that

$$F(D_1, \ldots, D_{27}) = F_a(A) + F_b(B)$$

where A and B are groups of D_i's such that no D_i is common to both A and B. A and B represent subproblems that can be solved separately. The general goal is to break the function into as many "factors" as possible such that there is no, or very little, interaction between factors.

The next step is to determine the interaction between all pairs of control variables, D_i and D_j, for example. Does the behavior of D_i with respect to the performance function F depend on D_j? Let $R_{(i,j)}$ be the answer where $R_{(i,j)}$ may take on one of four values as follows:

0 = no obvious dependency
1 = weak dependency
2 = moderate dependency
3 = strong dependency.

Based on White's subjective judgment, values for $R_{(i,j)}$ were generated and appear in Fig. 17. It is assumed that $R_{(i,j)} = R_{(j,i)}$. Note that the ordering of the items in the matrix produced very scattered data. The eye is not able to identify any striking organizational structure.

The Bond Energy Algorithm was applied using the original matrix as a starting point with the objective of rearranging the rows and columns of the matrix to obtain a better order. The algorithm tends to push the larger numbers together into clumps and favors large clumps over smaller ones. There is no preferential orientation of the final clumps; however, the symmetry of the original matrix about its diagonal results in a symmetrical final arrangement. The improved ordering is shown in Fig. 18. (The algorithm applied to the original matrix required about 2 minutes of CDC 1604 computer time for a number of starting points.)

After studying the matrix in Fig. 18, it appeared that there were eight clumps of numbers as indicated in the figure. The clumps contain all of the strong dependencies (the 3s) and all but six of the moderate dependencies (the 2s).

The interpretation of Fig. 18 is that clumps along the diagonal correspond to natural divisions of the big problem into subtasks. The off-diagonal elements not included in any clump correspond to coordination links. Figure 19 illustrates this interpretation. The items are

CONTROL VARIABLES

CONTROL VARIABLES	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
1	3	3	0	2	2	0	1	1	0	0	0	0	2	0	0	0	0	1	0	0	0	3	0	0	0	0	0
2	3	3	0	3	0	0	1	1	0	0	0	0	2	0	0	1	0	0	1	0	0	3	0	0	0	0	0
3	0	0	3	3	1	0	1	1	0	0	0	0	0	3	0	1	0	0	1	0	0	0	0	0	0	0	0
4	2	3	3	3	0	0	0	0	0	0	0	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0
5	2	0	1	0	3	0	2	2	1	1	0	0	0	0	0	3	0	0	1	0	3	1	0	0	0	0	0
6	0	0	0	0	0	3	0	0	2	1	0	0	0	0	0	0	1	0	0	1	0	0	3	0	0	1	2
7	1	1	1	0	2	0	3	2	2	3	1	0	1	1	0	1	0	0	0	0	0	1	0	0	0	0	0
8	1	1	1	0	2	0	2	3	2	2	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	1	2	2	2	3	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	1	1	3	2	3	3	1	1	2	2	0	0	0	0	1	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	1	0	0	1	3	3	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0
12	0	0	0	0	0	0	0	0	0	1	3	3	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0
13	2	2	0	1	0	0	1	0	0	2	0	1	3	0	0	1	0	0	1	0	0	2	0	0	0	0	0
14	0	0	3	1	0	0	1	0	0	2	0	1	0	3	0	1	0	0	1	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	1	1	1	0	0	3	1	0	0	0	0	1
16	0	1	1	0	3	0	1	1	0	0	0	0	1	1	1	3	1	1	1	0	3	2	0	0	0	0	0
17	0	0	0	1	0	1	0	0	0	0	0	0	0	0	1	1	3	0	0	1	1	0	1	0	0	0	3
18	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	3	0	0	0	1	0	0	2	0	0
19	0	1	1	0	1	0	0	0	0	1	0	1	1	1	0	1	0	0	3	0	0	1	0	0	0	0	0
20	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	1	0	0	3	2	0	0	3	0	1	0
21	0	0	0	0	3	0	0	0	0	0	0	0	0	0	3	3	1	0	0	2	3	0	0	1	0	1	1
22	3	3	0	0	1	0	1	0	0	0	0	0	2	0	1	2	0	1	1	0	0	3	0	0	0	0	0
23	0	0	0	0	0	3	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	3	0	0	0	1
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	1	0	0	3	0	2	1
25	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	2	0	0	0	0	0	0	3	0	3
26	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	2	0	3	0
27	0	0	0	0	0	2	0	0	0	0	0	0	0	0	1	0	3	0	0	0	1	0	1	1	3	0	3

(SEE TABLE 3 FOR DESCRIPTION OF VARIABLES BY NUMBER.)

11-12-69-18

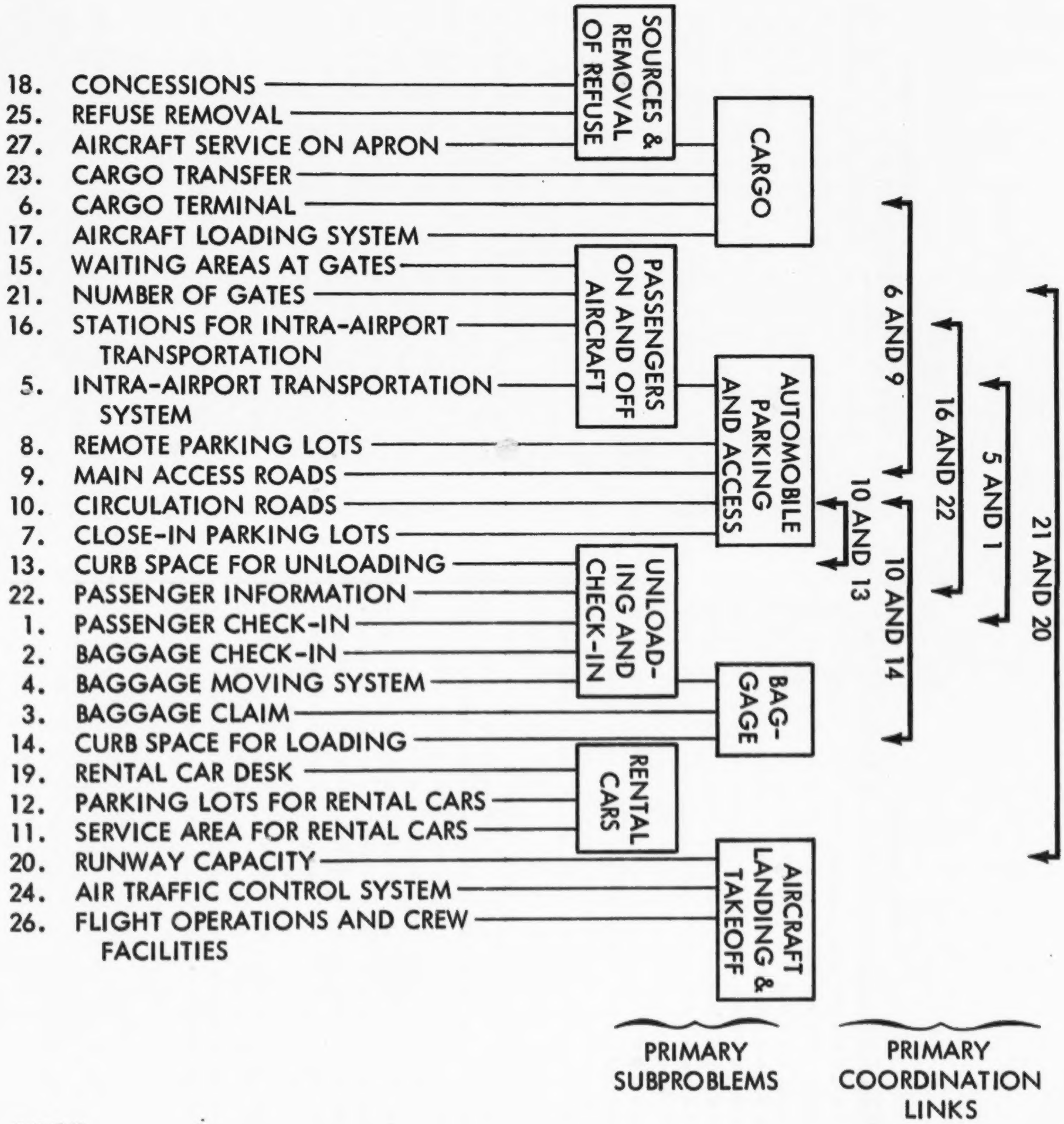
FIGURE 17. Dependency Matrix for Airport Variables

CONTROL VARIABLES

CONTROL VARIABLES	18	25	27	23	6	17	15	21	16	5	8	9	10	7	13	22	1	2	4	3	14	19	12	11	20	24	26
18	3	2	0	0	0	0	1	0	1	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0
25	2	3	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
27	0	3	3	1	2	3	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
23	0	0	1	3	3	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	2	3	3	1	0	0	0	0	0	$\boxed{2}$	1	0	0	0	0	0	0	0	0	0	0	0	1	0	1
17	0	0	3	1	1	3	1	1	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0
15	1	0	1	0	0	1	3	3	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
21	0	0	1	0	0	1	3	3	3	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	$\boxed{2}$	1	1
16	1	0	0	0	0	1	1	3	3	3	1	0	0	1	1	$\boxed{2}$	0	1	0	1	1	1	0	0	0	0	0
5	0	0	0	0	0	0	0	3	3	3	2	1	1	2	0	1	$\boxed{2}$	0	0	1	0	1	0	0	0	0	0
8	0	0	0	0	0	0	0	0	1	2	3	2	2	2	0	0	1	1	0	1	0	0	0	0	0	0	0
9	0	0	0	0	$\boxed{2}$	0	0	0	0	1	2	3	3	2	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	1	0	0	0	0	1	2	3	3	3	$\boxed{2}$	0	0	0	0	0	$\boxed{2}$	1	1	1	0	0	0
7	0	0	0	0	0	0	0	0	1	2	2	2	3	3	1	1	1	1	0	1	1	0	0	1	0	0	0
13	0	0	0	0	0	0	0	0	1	0	0	0	$\boxed{2}$	1	3	2	2	2	1	0	0	1	1	0	0	0	0
22	1	0	0	0	0	0	1	0	$\boxed{2}$	1	0	0	0	1	2	3	3	3	0	0	0	1	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	$\boxed{2}$	1	0	0	1	2	3	3	3	2	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	1	0	1	0	0	1	2	3	3	3	3	0	0	1	0	0	0	0	0
4	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	2	3	3	3	1	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	1	1	1	0	0	1	0	0	0	0	3	3	3	1	0	0	0	0	0
14	0	0	0	0	0	0	0	0	1	0	0	0	$\boxed{2}$	1	0	0	0	0	1	3	3	1	1	0	0	0	0
19	0	0	0	0	0	0	0	0	1	1	0	0	1	0	1	1	0	1	0	1	1	3	1	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	1	1	3	3	0	0	0
11	0	1	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	3	3	1	0	0
20	0	0	0	0	1	1	0	$\boxed{2}$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	3	3	1
24	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	3	3	2
26	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	2	3

11-12-69-19

FIGURE 18. Reordered Dependency Matrix for Maximum Clumpiness


18. CONCESSIONS
25. REFUSE REMOVAL
27. AIRCRAFT SERVICE ON APRON
23. CARGO TRANSFER
6. CARGO TERMINAL
17. AIRCRAFT LOADING SYSTEM
15. WAITING AREAS AT GATES
21. NUMBER OF GATES
16. STATIONS FOR INTRA-AIRPORT TRANSPORTATION
5. INTRA-AIRPORT TRANSPORTATION SYSTEM
8. REMOTE PARKING LOTS
9. MAIN ACCESS ROADS
10. CIRCULATION ROADS
7. CLOSE-IN PARKING LOTS
13. CURB SPACE FOR UNLOADING
22. PASSENGER INFORMATION
1. PASSENGER CHECK-IN
2. BAGGAGE CHECK-IN
4. BAGGAGE MOVING SYSTEM
3. BAGGAGE CLAIM
14. CURB SPACE FOR LOADING
19. RENTAL CAR DESK
12. PARKING LOTS FOR RENTAL CARS
11. SERVICE AREA FOR RENTAL CARS
20. RUNWAY CAPACITY
24. AIR TRAFFIC CONTROL SYSTEM
26. FLIGHT OPERATIONS AND CREW FACILITIES
SOURCES & REMOVAL OF REFUSE
CARGO
PASSENGERS ON AND OFF AIRCRAFT
AUTOMOBILE PARKING AND ACCESS
UNLOADING AND CHECK-IN
BAGGAGE
RENTAL CARS
AIRCRAFT LANDING & TAKEOFF
6 AND 9
10 AND 13
10 AND 14
16 AND 22
5 AND 1
21 AND 20
PRIMARY SUBPROBLEMS
PRIMARY COORDINATION LINKS

11-12-69-20

FIGURE 19. Suggested Subproblems with Coordination Links

listed along the left of Fig. 19 in the order found on the "clumped" matrix (Fig. 18). As a first approximation (shown below), the performance function can be split into eight factors corresponding to the eight subproblems shown in Fig. 19.

$$\begin{aligned} P \approx\ & F_a(D_{18}, D_{25}, D_{27}) \\ + \quad & F_b(D_{27}, D_{23}, D_6, D_{17}) \\ + \quad & F_c(D_{15}, D_{21}, D_{16}, D_5) \\ + \quad & F_d(D_5, D_8, D_9, D_{10}, D_7) \\ + \quad & F_e(D_{13,}, D_{22}, D_1, D_2, D_4) \\ + \quad & F_f(D_4, D_3, D_{14}) \\ + \quad & F_g(D_{12}, D_{11}) \\ + \quad & F_h(D_{20}, D_{24}, D_{26}) \end{aligned}$$

Except for D_{27} (aircraft service on apron) and D_5 (intra-airport transportation system), the eight components in the above approximation form independent subproblems. The six coordination links shown in Fig. 19 could form the basis of six "correction factors" which would improve the approximation. The correction factors would be of the form $\Delta_a(D_6, D_9)$, $\Delta_b(D_{10}, D_{13})$, $\Delta_c(D_{10}, D_{14})$, $\Delta_d(D_{16}, D_{22})$, $\Delta_e(D_5, D_1)$, and $\Delta_f(D_{20}, D_{21})$.

4. Ordering of Error Matrices in the Analysis of Perception of Consonants

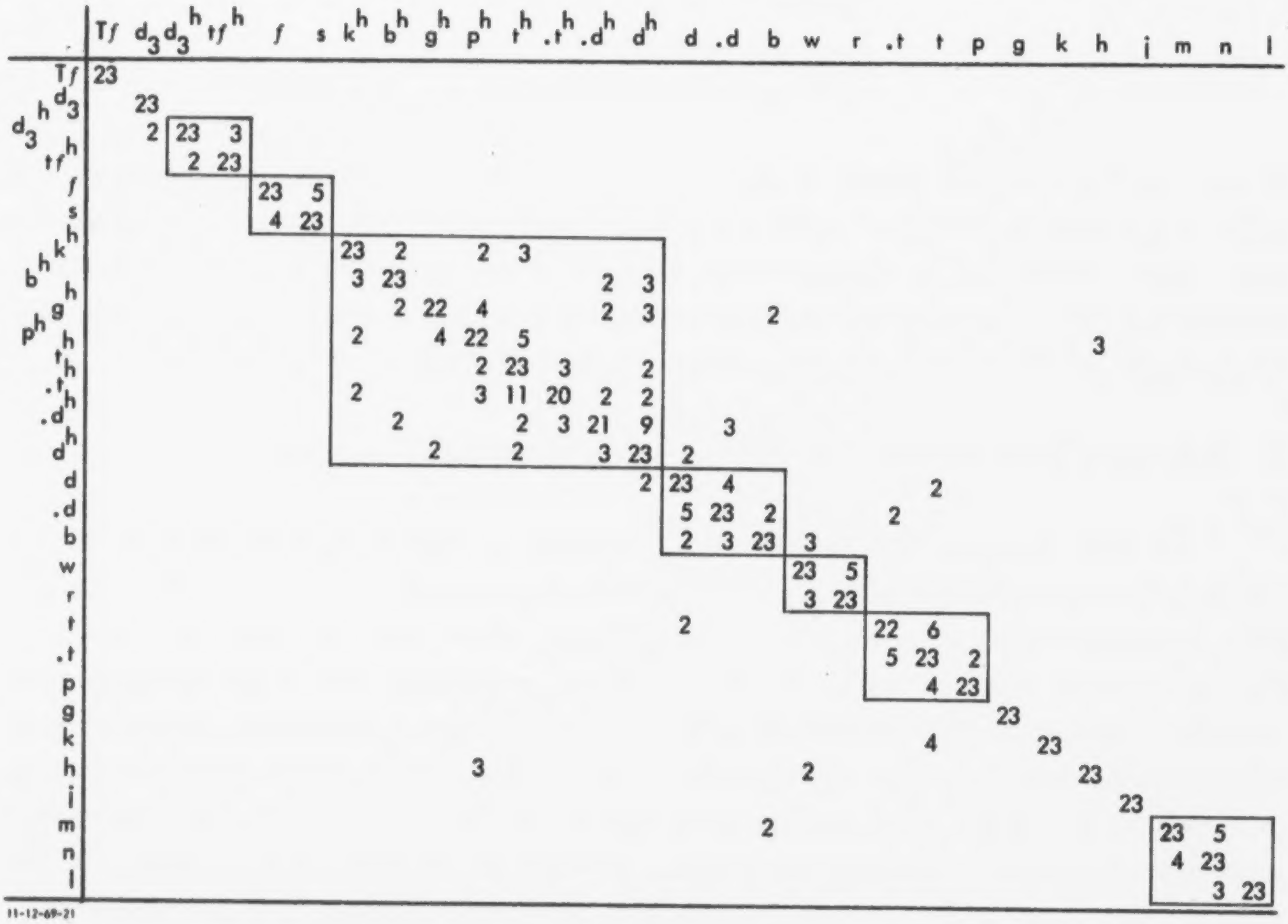
In this example, the Bond Energy Algorithm is used to reorder an error matrix obtained from an experiment testing the perception of consonants. The matrix under consideration is a square matrix with the 29 consonants lying on the vertical and horizontal axes. These data were taken from an article by Ahmed and Agrawal (Ref. 29) in the *Journal of the Acoustical Society*. In the experiment each consonant was enunciated in the intial position of 540 nonsense syllables. The α,β element of the matrix contained the number of times consonant β was heard when consonant α was spoken. It is clear than since the correct consonants are heard most often, the diagonal elements of the matrix will be largest. For this example, the square roots [14] of the elements were used rather than the elements themselves, and all the elements whose values are less than two are deleted.[15] The error matrix was input in a random manner and the best ordering of the reordered error matrix is shown in Fig. 20. The square blocks lying along the principal diagonal of the matrix indicate that the consonants have been clustered into 7 groups. These clusters represent those groups of consonants that were most often confused with one another during the experiment. The off-diagonal non-zero entries represent consonants in one group being mistaken for consonants outside their group. Note, for example, that one cluster contains the consonants d^h, d, .d, and b which occur together because they sound so much alike and hence were often mistaken for one another during the experiment.

---

[14] In this example the weighting k=2 is again used to preserve scale.

[15] The small elements are deleted so that the patterns formed by the larger elements may be visually identified more easily.

This example again illustrates how this method of direct analysis can be a significant aid in determining inherent group structure contained in data matrices.



	Tʃ	d_3	d_3^h	$tʃ^h$	ʃ	s	k^h	b^h	g^h	p^h	t^h	$.t^h$	$.d^h$	d^h	d	.d	b	w	r	.t	t	p	g	k	h	j	m	n	l
Tʃ	23																												
d_3		23																											
d_3^h		2	23	3																									
$tʃ^h$			2	23																									
ʃ					23	5																							
s					4	23																							
k^h							23	2		2	3																		
b^h							3	23					2	3															
g^h								2	22	4			2	3			2												
p^h							2		4	22	5														3				
t^h										2	23	3		2															
$.t^h$							2			3	11	20	2	2															
$.d^h$								2			2	3	21	9		3													
d^h									2		2		3	23	2														
d														2	23	4					2								
.d															5	23	2			2									
b															2	3	23	3											
w																		23	5										
r																		3	23										
t															2					22	6								
.t																				5	23	2							
p																					4	23							
g																							23						
k																					4			23					
h										3								2							23				
j																										23			
m																	2										23	5	
n																											4	23	
l																												3	23

FIGURE 20. Reordered (HINDI) Consonant Error Matrix

5. Inter-City Distances

In this example, the Bond Energy Algorithm is applied to a geographical problem to determine if it can satisfactorily cluster or clump together neighboring cities when a large number of inter-city distances are given as input data. Since the algorithm clusters large elements of a matrix, it was decided that the square root[16] of inverse distance would be used for the matrix elements. In particular, the elements of the matrix $a^{1/k}{}_{ij}$ are given by the expression, (with k = 2)

$$a^{1/2}{}_{ij} = \sqrt{\frac{100}{d_{ij}}} \qquad i \neq j, \text{ and}$$

$$a^{1/2}{}_{ij} \equiv 20 \qquad \text{all } i = j$$

where d_{ij} is the distance between city i and city j in hundreds of miles and the resulting matrix elements are rounded to the nearest integer. This input matrix is given in Fig. 21, where, for visual clarity, all the elements with values less than 7 have been deleted. It should be remembered from the definition that the larger the matrix element, the closer are the two cities i and j.

The matrix given in Fig. 22 is the reordered inverse distance matrix following operation by the algorithm on the input matrix. Elements whose values are less than 7 have again been deleted.

A number of clusters may easily be determined by identifying the square blocks of data that occur along the main diagonal of Fig. 22. The first two cities, Helena, Montana, and Bismark, North Dakota, are well isolated and constitute two separate clusters themselves. The next two cities, Denver, Colorado, and Cheyenne, Wyoming, are quite close and constitute a cluster. The next three cities, Des Moines, Iowa; Dubuque, Iowa; and Chicago, Illinois, are contained in the next cluster, and so forth. The one anomaly that does exist is the occurrence of the rectangular off-diagonal block of 7s. This indicates that although Chicago, Detroit, and Ft. Wayne are geographically near each other and are therefore in the same cluster, that Detroit and Ft. Wayne are also near some cities in another cluster, i.e., Cleveland, Akron, Columbus, and Cincinnati.

All these clusters may be verified geographically by referring to the map of the United States given in Fig. 23. The cities under consideration are denoted by darkened squares and the clusters are shown by the cities contained within each closed line.

---

[16] The square root was used to preserve scale and keep the matrix elements less than or equal to 20.

	AKRON	ATLANTA	BALTIMORE	BIRMINGHAM	BISMARK	BOSTON	BUFFALO	CHEYENNE	CHICAGO	CINCINNATI	CLEVELAND	COLUMBUS	DALLAS	DENVER	DES MOINES	DETROIT	DUBUQUE	EVANSVILLE	FT. WAYNE	HARRISBURG	HELENA
AKRON, O.	20						7			7	20	10				7			7		
ATLANTA, GA.		20		7																	
BALTIMORE, MD.			20																	10	
BIRMINGHAM, ALA.		7		20																	
BISMARK, N.D.					20																
BOSTON, MASS.						20															
BUFFALO, N.Y.	7						20				7										
CHEYENNE, WYO.								20						10							
CHICAGO, ILL.									20								7		7		
CINCINNATI, O.	7									20	7	10						7	7		
CLEVELAND, O.	20						7			7	20	10				7			7		
COLUMBUS, O.	10									10	10	20				7			7		
DALLAS, TEXAS													20								
DENVER, COLO.								10						20							
DES MOINES, IOWA															20		7				
DETROIT, MICH.	7										7	7				20			7		
DUBUQUE, IOWA									7					7			20				
EVANSVILLE, IND.										7								20			
FT. WAYNE, IND.	7								7	7	7	7			7				20		
HARRISBURG, PA.		10																		20	
HELENA, MONT.																					20

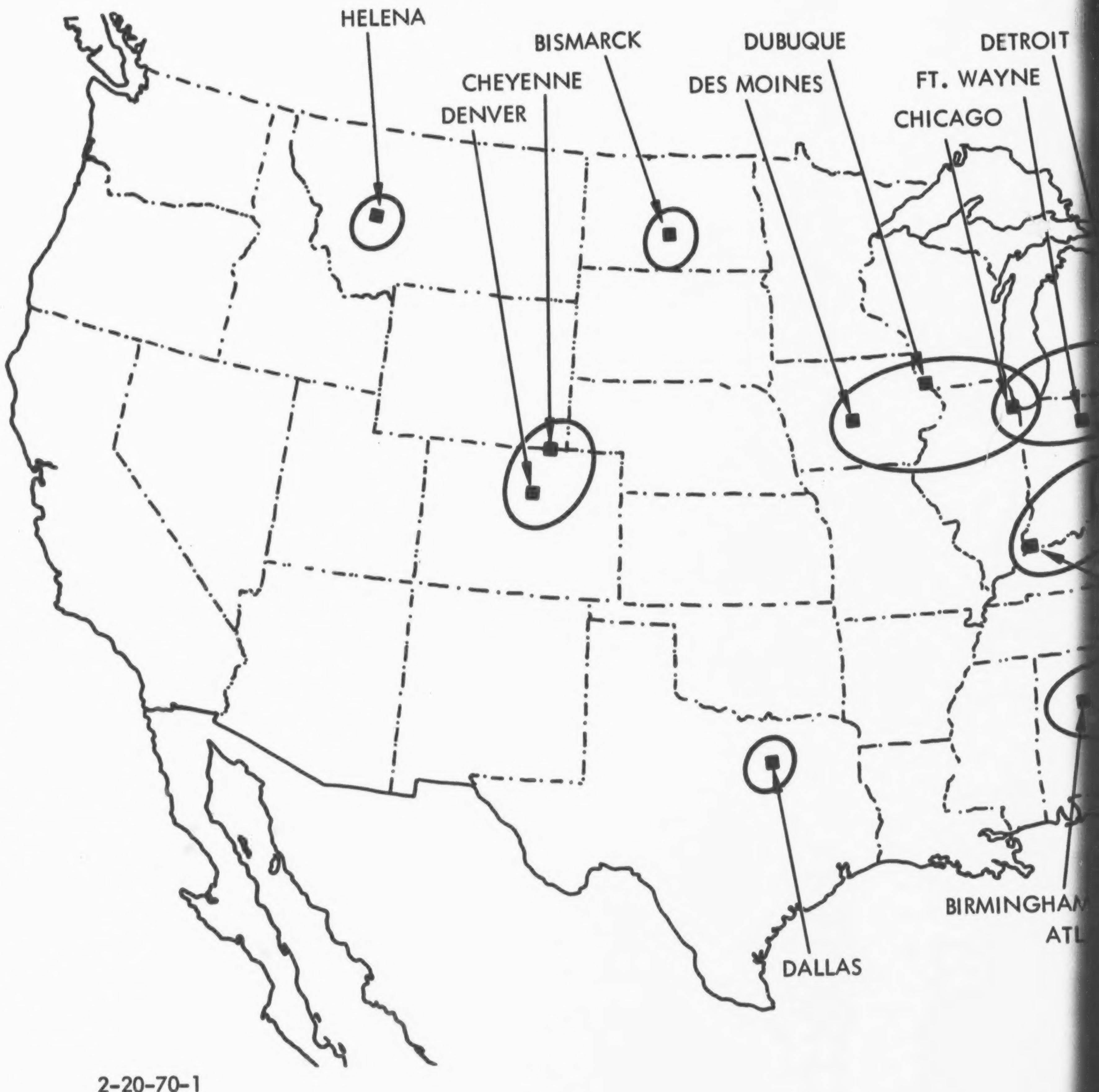
11-12-69-27

FIGURE 21. Initial Inter-City Inverse Distance Matrix

	HELENA	BISMARK	DENVER	CHEYENNE	DES MOINES	DUBUQUE	CHICAGO	DETROIT	FT. WAYNE	HARRISBURG	BALTIMORE	BOSTON	BUFFALO	CLEVELAND	AKRON	COLUMBUS	CINCINNATI	EVANSVILLE	ATLANTA	BIRMINGHAM	DALLAS
HELENA, MONT.	20																				
BISMARK, N.D.		20																			
DENVER, COLO.			20	10																	
CHEYENNE, WYO.			10	20																	
DES MOINES, IOWA					20	7															
DUBUQUE, IOWA					7	20	7														
CHICAGO, ILL.						7	20		7												
DETROIT, MICH.								20	7					7	7	7					
FT. WAYNE, IND.							7	7	20					7	7	7	7				
HARRISBURG, PA.										20	10										
BALTIMORE, MD.										10	20										
BOSTON, MASS.												20									
BUFFALO, N.Y.													20	7	7						
CLEVELAND, O.								7	7				7	20	20	10	7				
AKRON, O.								7	7				7	20	20	10	7				
COLUMBUS, O.								7	7					10	10	20	10				
CINCINNATI, O.									7					7	7	10	20	7			
EVANSVILLE, IND.																	7	20			
ATLANTA, GA.																			20	7	
BIRMINGHAM, ALA.																			7	20	
DALLAS, TEXAS																					20

11-12-69-28

FIGURE 22. Reordered Inverse Distance Matrix


HELENA
BISMARCK
DUBUQUE
DETROIT
CHEYENNE
DES MOINES
FT. WAYNE
DENVER
CHICAGO
BIRMINGHAM
ATL
DALLAS

2-20-70-1

FIGURE 23

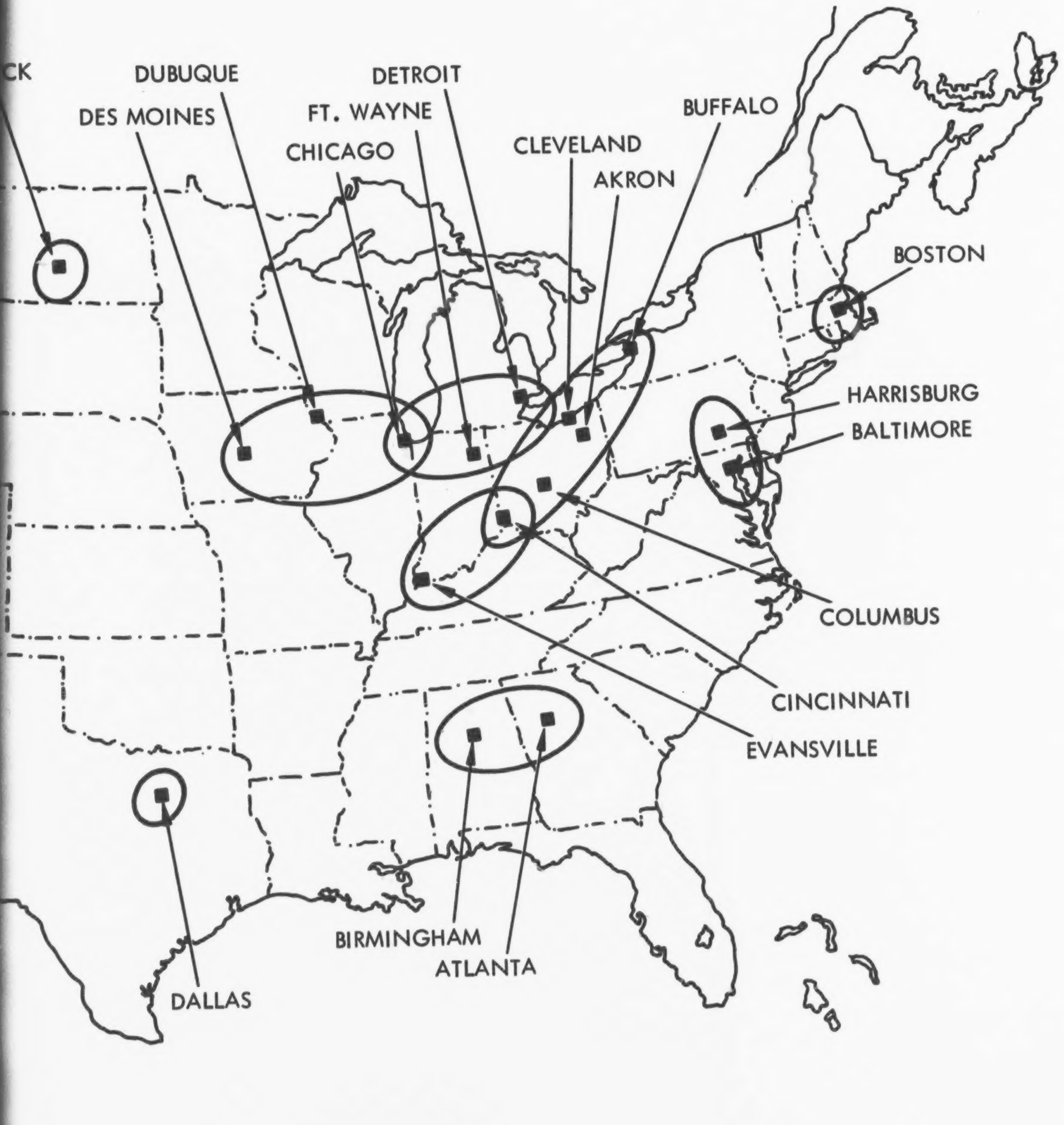

CK
DUBUQUE
DES MOINES
DETROIT
FT. WAYNE
CHICAGO
CLEVELAND
AKRON
BUFFALO
BOSTON
HARRISBURG
BALTIMORE
COLUMBUS
CINCINNATI
EVANSVILLE
BIRMINGHAM
ATLANTA
DALLAS

FIGURE 23. Geographical Illustration of City Clusters

The conclusion that can be drawn from this example is that the Bond Energy Algorithm can indeed rearrange data geographically when it is presented in another order (alphabetically).

II. THE MOMENT ORDERING ALGORITHM

A. INTRODUCTION

The purpose of the Moment Ordering Algorithm is to use the information contained in an array of data to find a one-dimensional ordering of the rows (and columns) of the array. This one-dimensional ordering will represent the ranking of the rows (and columns) under the relationship which the algorithm finds to be the most important in analyzing the array. The algorithm therefore provides a method of extracting, from the complex interrelationships which may be expressed in the array, a single important relationship, and of organizing the rows and columns according to this relationship. For example, one of the problems discussed below involves an array describing the voting pattern of Senators. The algorithm in this case takes the array, originally in the arbitrary form of an alphabetical listing of Senators and a chronological listing of votes, and produces an ordering of the Senators, and of the bills voted upon, based solely upon the original array, which represents a liberal/conservative ordering. A second example involves an array consisting of archeological sites as the columns, and of pottery types as the rows, with the entries being the concentration of a pottery type in a site. The algorithm in this case privides a reordering which puts the pottery types, and the sites, in a choronological order, based upon the fact that the most important factor in determining the types of pottery found at these sites was the age of the site.

B. THE ALGORITHM

1. Motivation

The definition of the algorithm is based upon the fact that if two rows are similar to each other, their mean row moments should be close to each other in value. The mean row moment x_i of row i is defined as

$$x_i = \frac{\sum_{j=1}^{N} j \; a_{ij}}{\sum_{j=1}^{N} a_{ij}} ,$$

where a_{ij} is the ij^{th} entry in the array. This is merely another way of stating that rows are similar if their large entries occur in the same columns, or in columns close to each other. Similarly, if two columns are closely related, their mean column moments, defined as

$$y_j = \frac{\sum_{i=1}^{M} i \ a_{ij}}{\sum_{i=1}^{M} a_{ij}}$$

should be close to each other in value.

Based upon these observations, then, it is desirable to arrange an array so that its rows are in order of the values of their row moments, while at the same time its columns are in order of the values of the column moments. This state will correspond to a one-dimensional ranking of both the rows and the columns according to the same underlying variable. The algorithm provides a method of finding such states, and hence of ordering arrays of data.

2. Definition

The algorithm, beginning with an arbitrary arrangement of an array, proceeds in the following way to find a state with the property described above, of having both the rows and the columns of the array arranged in order of their moments:

1. The row moments are calculated for the original arrangement of the array, and the rows are reordered to put them in order of their moments.

2. The column moments are calculated, and the columns reordered according to their moments.

3. Because the reordering of the columns changes the values of the row moments, the rows will no longer necessarily be in order of their row moments. The row moments are therefore recalculated for the new arrangement of the columns, and the rows reordered according to these new moments.

4. The procedure is continued, alternately reordering the rows and columns, until a state is found in which both are simultaneously in order of their moments. This state, then, is the desired ordering of the rows and columns, and is a solution of the algorithm.

The algorithm is therefore entirely an iterative procedure. The progress of the algorithm toward convergence, however, is marked by an increasing concentration of the larger elements on or near the main diagonal.[1]

The progress of the algorithm is illustrated, for a 4x4 array, in Fig. 24. The initial state of the array is a; the values of the row moments for that array arrangement are also shown. The algorithm then proceeds through states b, c, and d, by reordering the rows and columns alternately. When state e is reached, it is found that the rows are already in the proper order and do not need to be reordered. This marks that state as a solution.

The concentration of the larger elements on or near the main diagonal in the solution is pointed out in Fig. 24 by circling, in the initial and final states, the four largest elements. They are scattered in the initial state but in the solution three are on the main diagonal and one is just off it.

The following subsections present further details concerning the use of the algorithm. Section C presents several specific problems which have been investigated by use of the algorithm, and illustrates the utility of the orderings produced by the algorithm.

3. Stable States and Multiple Solutions

The algorithm as defined above takes an arbitrary initial ordering of an array and finds a stable reordering. It has been found, however, that if different initial orderings of the same array are used, different solutions may be found. For example, Fig. 25 shows two different solutions which can be found for a simple 3x3 array.[2]

When the algorithm is run many times on larger arrays, using different starting orderings each time, it has been found that those solutions which occur most frequently always are amongst the most diagonal arrangements of the array.[3] Conversely, any solutions which are very nondiagonal occur only rarely.

This observation has been used as the basis of a technique for obtaining a final ordering of the rows and columns which best utilizes the additional information found in the multiple solutions.

---

[1] Appendix H discusses a measure of effectiveness which has been defined to measure this progress toward diagonality. However, because unlike the Bond Energy Algorithm this algorithm was not developed to maximize this quantity, the measure of effectiveness defined has been found to be of only marginal use.

[2] Appendix I describes an investigation which was made, for a 3x3 array, of the properties which lead to the existence of these multiple solutions.

[3] As measured by the correlation coefficient measure of effectiveness defined in Appendix H.

(INITIAL STATE)

(a)

	α	β	γ	δ	ROW MOMENTS
A	3	3	4	5	2.73
B	⑧	4	5	2	2.05
C	2	1	⑧	3	2.86
D	⑦	⑥	2	5	2.25

REORDER ROWS ↓

(b)

	α	β	γ	δ
B	8	4	5	2
D	7	6	2	5
A	3	3	4	5
C	2	1	8	3
NEW COLUMN MOMENTS	1.95	2.07	2.79	2.60

REORDER COLUMNS ↓

(c)

	α	β	δ	γ	NEW ROW MOMENTS
B	8	4	2	5	2.21
D	7	6	5	2	2.10
A	3	3	5	4	2.67
C	2	1	3	8	3.21

REORDER ROWS ↓

(d)

	α	β	δ	γ
D	7	6	5	2
B	8	4	2	5
A	3	3	5	4
C	2	1	3	8
NEW COLUMN MOMENTS	2.00	1.93	2.40	2.95

REORDER COLUMNS ↓

(e)

	β	α	δ	γ	NEW ROW MOMENTS
D	⑥	⑦	5	2	2.15
B	4	⑧	2	5	2.42
A	3	3	5	4	2.67
C	1	2	3	⑧	3.29

NO REORDERING OF
ROWS NECESSARY
(SOLUTION)

11-13-69-1

FIGURE 24. Operation of the Algorithm on a Small Array

SOLUTION 1

	α	β	γ
A	7	2	1
B	3	5	2
C	3	0	7

SOLUTION 2

	α	γ	β
A	7	1	2
C	3	7	0
B	3	2	5

11-13-69-3

FIGURE 25. Illustration of Multiple Solutions

The algorithm is run a "large" (25 or 50 has been found satisfactory) number of times, each time starting from a different random ordering of the rows or columns, and the order of the rows and columns found each time is saved.[4] The average of the position taken by each row (and column) in the solutions is found. (Solutions found more than once are entered once for each time found in obtaining the average.) The rows' and columns' final order is then simply the order of their average positions. Most often, this order will be the same as the order in the most common solution; it is always very close to that order.

Despite the additional complication introduced, this technique is considered preferable to merely taking the most common solution, because in the event that several solutions are common, this technique best takes into account the alternative orderings each solution represents in arriving at a consensus final ordering.

4. Additional Details

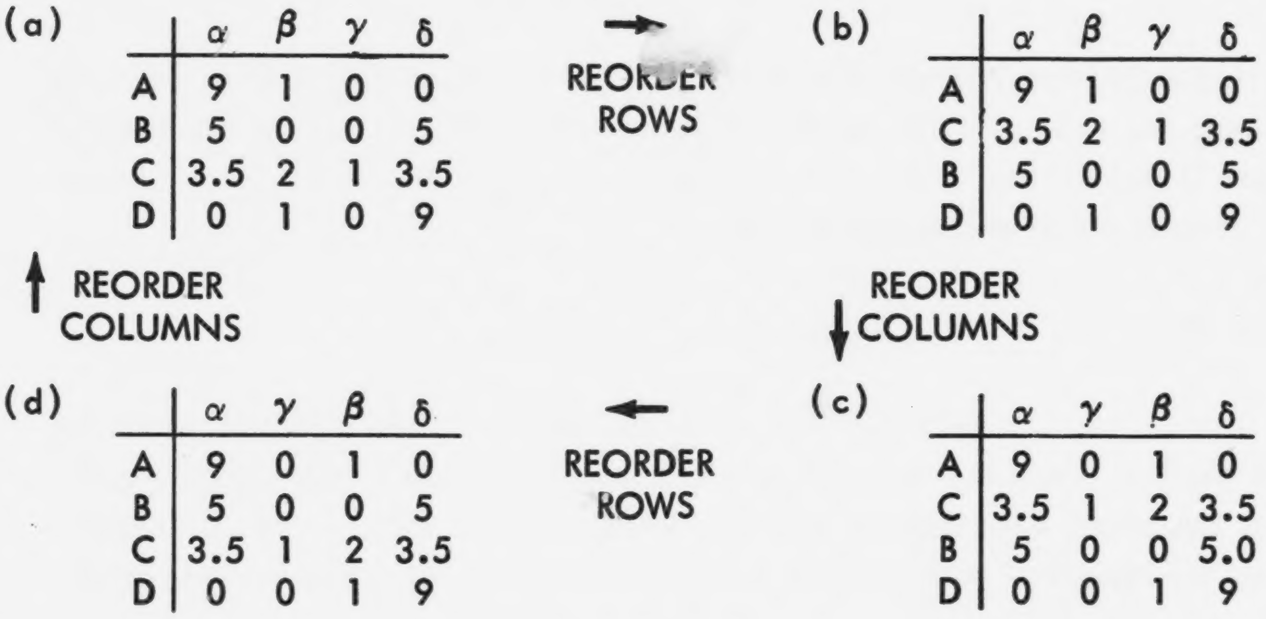
The previous sections have discussed all of the features of the algorithm which are important in practice. There are, however, two points of theoretical interest which must be mentioned at this point. Both concern situations which can arise in the process of iteration carried out by the algorithm. Both occur so rarely, however, that in practice they can usually be ignored.

a. Ties. In carrying out the algorithm, two or more rows or columns may have identical moments. In this case it is necessary to resolve the tie to obtain an ordering so that the algorithm can proceed. This is done by trying all permutations of non-identical rows (or columns) and selecting that particular row (or column) order which yields the highest value of the correlation coefficient R.[5] If several permutations of the rows (or columns) have the same value of R, the algorithm simply accepts the last order investigated. It should be noted that ties, while prominent for small, binary (0-1), arrays, very rarely occur when dealing with large arrays containing non-binary data.

---

[4] Note that a particular order and its reverse are considered identical and saved as the same order.

b. Cycling. According to the definition of the algorithm, the iterative procedure is continued until a stable state unchanged by either row or column operations (Fig. 24, for example) is found. In fact, however, it is theoretically possible that, instead of arriving at such a stable state, the algorithm may cycle between a small set of states. Fig. 26 illustrates the phenomenon for a specially designed small array (in actual fact such cycling has only been found in very much larger arrays).[6] Once the algorithm arrives at the state shown in Fig. 26, which it can reach from many other states, it will cycle forever between a, b, c, and d, in that order. Such an "infinite loop" itself represents a final state of the array. The procedure used when such cycling is detected therefore is to terminate the iterations and take one of the states involved in the loop as the solution.



(a)

	α	β	γ	δ
A	9	1	0	0
B	5	0	0	5
C	3.5	2	1	3.5
D	0	1	0	9

(b)

	α	β	γ	δ
A	9	1	0	0
C	3.5	2	1	3.5
B	5	0	0	5
D	0	1	0	9

(c)

	α	γ	β	δ
A	9	0	1	0
C	3.5	1	2	3.5
B	5	0	0	5.0
D	0	0	1	9

(d)

	α	γ	β	δ
A	9	0	1	0
B	5	0	0	5
C	3.5	1	2	3.5
D	0	0	1	9

11-13-69-2

FIGURE 26. Illustration of Cycling Phenomenon

In practice, this phenomenon has been observed only very rarely, and only in very large arrays. Furthermore, even when it does occur, it has been found that most often the algorithm will find normal stable solutions when operating upon the same array from other starting points. For this reason, this cycling, while theoretically quite objectionable, has been found to be of little operational difficulty.

---

[5] See Appendix H.

[6] The symmetry and normalization inherent in the array of Fig. 26 are not necessary for the cycling to occur, but were built in to simplify the array.

γ

5. Computation Time

For an MxN matrix, M operations are required to compute each column moment and N operations to compute each row moment. Therefore, for each iteration, the total number of operations necessary to reorder the rows and columns is 2MN. Finally, if it requires I iterations for the algorithm to converge, the total number of operations to reach a solution for each random starting point is 2IMN. The computer time required on the CDC 1604 to solve a particular 29x29 matrix was about 24 seconds for one starting point; an 80x80 matrix took 4 minutes. Note that these times are influenced by the number of iterations required for convergence as well as by the matrix sizes.

C. RESULTS

This section describes two problems investigated with the Moment Ordering Algorithm. It demonstrates that the algorithm can in fact uncover a dominant relationship from the vast amount of information in a matrix, and can produce orderings of the rows and columns which reflect this relationship.

1. U.S. Senate Voting Patterns

The algorithm was used to study the relationships between the voting patterns of a group of U.S. Senators. The hope was that, given only the recorded positions of Senators on a random group of issues, the algorithm could generate a meaningful ordering. The first 20 Senators (alphabetically) in 1968 were chosen, and their recorded positions[7] on 12 issues were tabulated (see Tables 4 and 5). The recorded position of the President on each issue was added to the table, and the algorithm was applied to the resulting 21x12 array. The results, as shown in Table 6, showed an ordering from conservative Republican and Southern Democrat at one end to liberal Democrat on the other. To be sure that the strong ordering was not an accident, the same type of array was constructed for 12 different roll calls (but the same Senators), and the algorithm was rerun. The correlation between the two sets of results (see Table 6 again) indicates that the ordering found was significant. The difference between the two rankings does not, it is emphasized, reflect any inherent limitation upon the accuracy of the algorithm, but rather is a result of the limited sizes of the samples of votes used in the analyses. If more roll calls were added to the arrays, the results would approach each other more and more, reflecting the enlarged and therefore improved sampling. The algorithm's solution indicates that, as might be expected, although a Senator's position on any given issue may not always be

---

[7] As taken from tables in *Congessional Quarterly Almanac*, Vol. 24, 1968, pp. lS-58S.

Table 4. *SENATE ROLL CALL VOTES INCLUDED IN ANALYSIS* [a]

		ROLL CALL[b]																							
	SENATOR[c]	ARRAY 1												ARRAY 2											
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
1	President	0	0	1	0	.5	.5	1	0	0	1	1	1	.5	.5	.5	0	0	0	.5	1	.5	1	.5	.5
2	AIKEN	0	0	1	0	0	0	1	1	1	1	1	0	0	1	0	0	0	0	1	1	0	1	0	0
3	ALLOTT	0	1	1	1	1	1	1	0	.5	0	1	0	0	1	1	0	1	0	1	1	0	1	.5	0
4	Anderson	0	1	1	0	1	1	0	.5	0	0	1	1	0	.5	.5	0	0	0	0	0	1	1	.5	0
5	BAKER	0	1	1	1	0	.5	1	.5	.5	0	1	0	0	1	1	0	1	0	0	1	1	1	0	0
6	Bartlett	0	1	1	0	0	0	0	1	1	0	1	1	1	0	1	0	0	1	.5	0	1	1	1	.5
7	Bayh	0	1	1	0	0	1	0	1	0	0	1	1	0	0	0	1	0	1	1	0	1	1	.5	0
8	BENNETT	1	1	1	0	1	1	1	0	1	0	0	0	1	1	1	0	1	0	0	1	0	0	0	0
9	Bible	0	.5	1	0	0	1	0	1	0	0	1	1	.5	1	1	0	.5	0	1	0	1	0	.5	0
10	BOGGS	0	1	1	0	0	0	1	0	1	0	1	0	0	1	1	0	1	1	1	1	0	1	0	1
11	Brewster	0	1	1	0	0	0	0	0	0	1	1	1	0	0	0	0	0	1	1	0	1	1	.5	0
12	BROOKE	0	0	1	0	1	0	1	1	1	1	1	1	0	1	0	1	0	0	0	1	0	1	0	1
13	Burdick	0	0	1	0	1	0	0	1	0	0	1	1	0	0	0	1	0	1	1	0	1	0	1	0
14	Byrd, Jr.	1	1	0	0	0	1	1	0	1	0	0	0	1	1	1	0	1	1	0	1	0	0	0	0
15	Byrd	1	1	0	0	0	1	0	1	1	1	0	1	1	0	1	0	0	1	0	0	1	0	0	0
16	Cannon	0	1	1	0	1	1	0	0	0	0	0	1	1	1	1	0	0	0	1	0	1	1	1	0
17	CARLSON	1	1	1	.5	0	1	1	0	1	0	1	0	0	1	1	0	1	0	0	1	.5	.5	0	0
18	CASE	0	0	1	0	0	0	1	1	0	1	1	1	0	1	0	0	0	0	1	0	1	1	0	1
19	Church	0	1	1	0	1	0	0	.5	0	0	1	1	0	0	1	1	0	1	1	0	0	0	0	0
20	Clark	0	0	1	0	0	0	0	1	0	1	1	1	0	0	0	1	0	1	1	0	1	1	1	1
21	COOPER	1	0	1	0	0	0	1	0	1	1	1	0	.5	0	1	1	1	1	0	1	.5	1	0	1

[a] Information from *Congressional Quarterly Almanac,* Vol. 24, 1968, pp. 15S-58S.
1 = a position in favor; .5 = no recorded position; 0 = opposition.

[b] Subjects of roll calls identified in Table 5.

[c] Republicans in capital letters.

Table 5. ARRAYS USED IN SENATE VOTE PATTERN ANALYSIS[a]

Vote No.	Roll Call[b]	Vote	Subject Matter	Sponsor
1	10	33 - 58	Amendment to open housing bill to bar federal courts from impairing title to real property as recorded under state recording statutes.	Ervin
2	20	62 - 21	Amendment to open housing bill to punish anyone instructing in the use of firearms for riots, or interfering with police during a riot.	Long
3	30	61 - 19	Amendment to open housing bill to provide a compromise bill.	Dirksen
4	40	19 - 58	Amendment to gold cover removal bill to limit expansion of Federal Reserve notes in circulation to 4% per year.	Allott
5	50	43 - 28	Amendment to Standards of Conduct Resolution to allow use of political contributions for certain office expenses.	Yarborough-Javits
6	60	38 - 44	Amendment to excise tax extension bill to provide 20% surtax on people trading with Communist nations which supply North Vietnam.	Mundt-Byrd (Va.)
7	70	53 - 35	Amendment to excise tax extension bill to impose 10% income tax surcharge and limit expenditures to $180 billion.	Williams-Smathers
8	80	28 - 30	Amendment to Military Procurement Authorization to cut R&D funds from $7.9 to $7.4 billion.	Hart
9	90	39 - 29	Amendment to Conservation Fund bill to remove outer continental shelf revenues from fund for 1972 and 1973.	Williams
10	100	29 - 53	Amendment to Omnibus Crime Bill to prohibit interstate mail order sales of rifles and shotguns.	Kennedy
11	110	51 - 30	Amendment to Omnibus Crime Bill to delete language denying Supreme Court jurisdiction to review state court judges' decisions to admit eyewitness testimony in evidence.	Tydings
12	120	33 - 44	Amendment to Omnibus Crime Bill to allocate 2/3 instead of 85% of funds in block grants to states.	Brooke
13	127[c]	25 - 35	Amendment to require cities as well as states to reimburse NIDC for costs of riot losses insured by NIDC.	Russell
14	137[c]	42 - 27	Amendment to delete section on retirement benefits from bill to extend term of office of bankruptcy referees.	Carlson
15	150	44 - 32	Motion to table amendment which would have provided $52 million supplemental appropriation to Labor Department for summer jobs.	Holland
16	160	16 - 61	Amendments to Military Construction Authorization to cut Navy and Air Force funds by 10%.	Clark
17	170	34 - 38	Amendment to juvenile delinquency bill to allocate all funds as block grants to states.	Murphy
18	180	34 - 52	Amendment to Federal Agency Authorization to cut NASA R&D funds an additional $300 million.	Williams
19	190	30 - 40	Amendment to Agricultural Act to limit to $75,000 payments to one producer for participation in certain agricultural programs.	Williams
20	200	46 - 45	Amendment to Interest Rates Bill to strike out language authorizing Federal Reserve banks to purchase obligations directly from federal agencies.	Bennett
21	211[c]	51 - 22	Amendment to strike out language added by House which limited expenditures of State, Justice, and Commerce to $1.98 billion.	Committee
22	221[c]	46 - 28	Foreign Aid Authorization Bill.	—
23	230	23 - 35	Amendment to Renegotiation Act to exempt Renegotiation Board from employee limitations.	Proxmire
24	240	31 - 53	Amendment to Gun Control Act to add a registration provision.	Brooke

[a] Information is taken from the *Congressional Quarterly Almanac*, (Vol. 24, 1968), pp. 1S-58S.

[b] The first 12 roll calls above are included in the first array, the second 12 in the second array.

[c] When a roll call selected was too one-sided to convey significant information, a roll call close in time to it was substituted.

Table 6. RESULTS OF VOTE PATTERN ANALYSIS[a]

	Order	Array 1[b]	Array 2[b]
	1	Burdick	Burdick
↕ CONSERVATIVE — LIBERAL	2	Clark	Bayh
	3	Church	Clark
	4	Brewster	Brewster
	5	Bible	Church
	6	CASE	Bartlett
	7	Bayh	Byrd (W.Va.)
	8	Anderson	Anderson
	9	Bartlett	CASE
	10	President	Cannon
	11	BROOKE	COOPER
	12	Cannon	BROOKE
	13	AIKEN	Bible
	14	Byrd (W.Va.)	President
	15	ALLOTT	BOGGS
	16	BAKER	AIKEN
	17	BOGGS	ALLOTT
	18	COOPER	Byrd (Va.)
	19	BENNETT	BAKER
	20	CARLSON	CARLSON
	21	Byrd (Va.)	BENNETT

[a] See Tables 4 and 5 for input data.

[b] Republicans in capital letters.

predictable, overall voting patterns based upon ideology are strongly evident, and Senators can be placed reasonably well on a liberal-conservative spectrum. More important, for our purposes, it indicates that when a meaningful ordering is inherent in a set of data, the algorithm will find that ordering.

2. Chronological Ordering in Archaeology

The algorithm was used to attempt to order a series of archaeological deposits. The basic data available is the distribution of various types of pottery (eight, in this case) among various deposits of archaeological interest (also eight, in this case). Robinson (Ref. 30), upon whose work this example is based, hypothesized that it should be possible to arrange these sites into a proper chronological order by assuming that pottery types come into and go out of general use in a regular manner over time, and that, therefore, deposits similar to each other in the amounts of various types of pottery will be close to each other in time as well. Thus, if a satisfactory one-dimensional arrangement of the pottery deposits can be found, on the basis of a pottery-type percentage array, the sites should be chronologically ordered. This was therefore used as a test of the Moment Ordering Algorithm.

The raw data matrix presented by Robinson in Ref. 30 is shown in Table 7. If the algorithm is performed on this array, the solution found is 3A, 2A, 3B, 1A, 3C, 2B, 1B, 2C, which is very close to that presented by Robinson, and which satisfies the tests he carries out on his candidate solution.

Table 7. RAW POTTERY PERCENTAGES

Pottery Type	Deposit							
	2A	2B	2C	1A	1B	3A	3B	3C
1	24.0	1.4	0.2	11.3	0.3	29.6	54.3	0
2	66.8	0.9	0	0	0	0	3.5	0
3	1.3	0	0.2	3.8	0.2	14.1	14.0	6.6
4	0	0	0	1.3	0.2	0	1.8	3.3
5	0	0	0	3.3	0.5	0	5.3	5.5
6	4.0	0	0	24.9	1.4	7.0	7.0	27.5
7	0	97.7	99.3	52.6	97.4	0	12.3	57.1
8	3.9	0	0.3	2.8	0	49.3	1.8	0

Robinson, however, introduces an "agreement coefficient" between two pottery types, defined arbitrarily as:

$$a_{ij} = 200 - \sum_{k=1}^{N} | P_{ik} - P_{jk} | \text{ , where there are N sites,}$$

and P_{ik} and P_{jk} are the *percentages* of types i and j in site k. Therefore, $a_{ij} = 200$ constitutes total agreement between the composition of two sites, $a_{ij} = 0$, total disagreement. Robinson's resulting array is presented in Table 8. Robinson then attempts to carry out a "rearrangement" of this array to drive large numbers toward the diagonal; he describes a semi-systematic manual method of doing so and presents the resulting order as his solution. The Moment Ordering Algorithm was run on Table 8 and found exactly the same order as Robinson's method–2A, 3A, 3B, 1A, 3C, 1B, 2B, 2C. The reordered matrix of agreement coefficients is shown in Table 9, where it is apparent that the larger matrix elements have accumulated around the main diagonal of the array. The advantage obtained in using the algorithm, of course, lies in the fact that it is an automatic, systematic approach and does not require personal judgments to be made, as Robinson's method did. The fact that it reproduces Robinson's chronological ordering reinforces the belief that the algorithm is suitable for just this problem–ordering entities in one dimension based on their interrelationships.

Table 8. AGREEMENT COEFFICIENTS

Pottery Deposit	Pottery Deposit							
	2A	2B	2C	1A	1B	3A	3B	3C
2A	200	5	1	39	4	66	69	11
2B	5	200	196	108	195	3	29	114
2C	1	196	200	107	196	1	26	115
1A	39	108	107	200	110	50	82	172
1B	4	195	196	110	200	4	30	119
3A	66	3	1	50	4	200	101	27
3B	69	29	26	82	30	101	200	66
3C	11	114	115	172	119	27	66	200

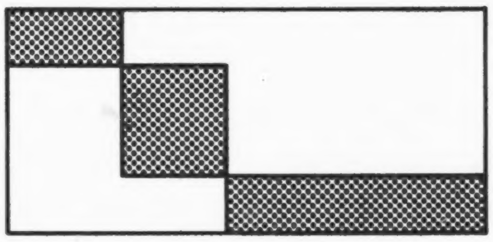
Table 9. REORDERED AGREEMENT COEFFICIENTS

Pottery Deposit	Pottery Deposit							
	2A	3A	3B	1A	3C	1B	2B	2C
2A	200	66	69	39	11	4	5	1
3A	66	200	101	50	27	4	3	1
3B	69	101	200	82	66	30	29	26
1A	39	50	82	200	172	110	108	107
3C	11	27	66	172	200	119	114	115
1B	4	4	30	110	119	200	195	196
2B	5	3	29	108	114	195	200	196
2C	1	1	26	107	115	196	196	200

III. THE MOMENT COMPRESSION ALGORITHM

A. INTRODUCTION

The Moment Compression Algorithm discussed in this chapter is based upon the key observation that the distinguishing feature of a matrix in perfect block form, (see sketch) when contrasted with the same matrix after row or column permutations, is that the moment of inertia of each row and column about its mean is minimized: any row or column permutation of a matrix in perfect diagonal block form will "expand" a block and make it less dense, thereby increasing the matrix's summed moments of inertia.

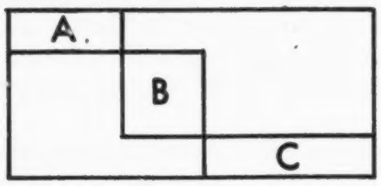


Consequently a procedure which minimizes, by row and column permutations, the sums of the row and column mean square moments about their means will drive the matrix into perfect block form if this is possible.[1,2] If this is not possible, the procedure will still tend to produce a pleasing pattern because it tries to create dense blocks. This reasoning led to the development of the Moment Compression Algorithm.

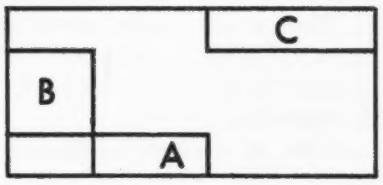
Although Moment Compression has been superseded by Bond Energy both as a theoretical ME and as a computational procedure, this material is being presented both to indicate an approach which was explored and found impractical, and to show a logical stepping-stone in the development of the Bond Energy Algorithm. Moment Compression was historically important for four reasons:

---

[1] Ambiguity will still exist because



and



will be considered equally good. But one would be indifferent to such ambiguity as long as the variables have been factored correctly.

[2] This assertion is proved in Appendix C.

(1) It was our first attempt[3] to describe the appeal of a pattern in terms of a quantitative ME, the sums of the moments of inertia. This was motivated by a desire to produce dense blocks of numbers.

(2) It was our first attempt to devise an algorithm based on ME-optimization. This was in contrast to heuristic algorithms, such as moment ordering and some similarity matrix approaches, where it was not clear what each step in the algorithm was trying to accomplish. In particular, rigorous optimization of the ME would avoid the problems of cycling and non-uniqueness[4] experienced in the Moment Ordering Algorithm.

(3) It was our first attempt to devise algorithms which find near-optimal, rather than optimal, solutions for the ME. The major pitfall encountered in the Moment Compression case, but not in the Bond Energy case, was that the approximate algorithm was slow[5] and poor.[6]

(4) It used an ME which decomposed into two parts, one (sum of the row moments) dependent only on column permutations and the other (sum of the column moments) dependent only on row permutations. Consequently optimization of the ME could be achieved in exactly two passes, one finding the optimal column permutation, the other finding the optimal row permutation. These two passes are carried out completely independently of each other, and in particular, it is not necessary to alternate between row and column permutations, as in the Moment Ordering Algorithm. This decomposition of the ME into two parts was an attractive feature later used in the Bond Energy ME (row-bonds and column-bonds being optimized separately).

---

[3] Dr. Gould had earlier suggested use of the matrix correlation coefficient as a guide to the performance of the Moment Ordering Algorithm, but there was no particular pattern that one hoped to drive the matrix into.

[4] Cycling can never occur in an algorithm which iteratively optimizes an ME, for the ME is monotone from one iteration to the next. There would still be non-uniqueness if a few permutations achieved the global optimum; this could be expected only in degenerate cases, and normally would not occur. Permutations achieving local (rather than global) optima of the ME could be discarded on the basis of their inferior MEs, so that many fewer "stable" solutions could be expected than in the Moment Ordering Algorithm.

[5] At least a factor of three slower than the Bond Energy Algorithm, and therefore impractical for problems larger than about 25x25.

[6] While the algorithm is always successful at putting a matrix into near block diagonal form, if this is possible, it had two major weaknesses of (1) sensitivity of the result to the starting point, and (2) an inability to handle the checkerboard case, shown in Fig. 6.

B. MEASURE OF EFFECTIVENESS FOR MOMENT COMPRESSION

As stated above, the measure of effectiveness for Moment Compression is the sum of the mean-square[7] column moments and mean-square row moments. For any NxM non-negative matrix (b_{ij}), the ME is

$$ME(b) = \sum_{i=1}^{N} r_i + \sum_{j=1}^{M} c_j$$

where r_i is the row moment for the i^{th} row:

$$r_i(b) = r_i = \frac{\sum_{j=1}^{M} b_{ij} \left[j - \frac{\sum_{k=1}^{M} b_{ik} k}{\sum_{n=1}^{M} b_{in}} \right]^2}{\sum_{m=1}^{M} b_{im}}$$

and c_j is the column moment for the j^{th} column:

$$c_j = \frac{\sum_{i=1}^{N} b_{ij} \left[i - \frac{\sum_{k=1}^{N} b_{kj} k}{\sum_{n=1}^{N} b_{nj}} \right]^2}{\sum_{m=1}^{M} b_{mj}}$$

Let $A = \left[a_{ij}\right]$ be the original NxM non-negative matrix and let $\left[b_{ij}\right] = \left[a_{i,\pi(j)}\right]$ denote the matrix whose j^{th} column is the $\pi(j)^{th}$ column of A, where $\pi = \{\pi(1), \pi(2), \ldots, \pi(M)\}$ denotes a permutation of $\{1, 2, \ldots, M\}$. The problem of finding the best column permutation of A is given by

---

[7] While any even moment can be used, the second moment is the simplest.

$$\min_{\pi} \sum_{i=1}^{N} r_i(b)$$

$$= \min_{\pi} \sum_{jk=1}^{M} Q_{jk\pi(j)\pi(k)}$$

where

$$Q_{jkrs} = \sum_{i=1}^{N} \left[\frac{a_{ir} j^2 \delta_{jk}}{w_i} - \frac{a_{ir} a_{is} jk}{w_i^2} \right] \qquad 1 \leqslant jkrs \leqslant M$$

and $w_i = \sum_{j=1}^{M} a_{ij}$ denotes the row sum for the i^{th} row.

Finding the best row permutation leads to a problem completely analogous to that of finding the column permutation.

The above problem involves a minimization over all M! possible permutations. It is called a quadratic assignment problem because of the double appearance of π in the minimand. The problem of ME optimization is consequently equivalent to solving two quadratic assignment problems. (Appendix A demonstrates that the same holds true for the Bond Energy ME.)

As discussed in Appendix A, exact algorithms for solving quadratic assignment problems are too time consuming to be practical for M larger than 15 or 20. Consequently, an approximate algorithm was employed to find a near-optimal solution. The approximate algorithm is a gradient search in M^2-dimensional space, and is described in the next section.

C. GRADIENT ALGORITHM FOR APPROXIMATE ME-OPTIMIZATION

The minimization problem posed above can be rewritten as

$$\min_{X \epsilon PM} Z(X)$$

where PM denotes the set of all M! possible MxM permutation matrices $\left(\text{i.e., all matrices of the form } X_{ij} = \delta_{j,\pi(i)}\right)$ and where

$$Z(X) = \sum_{jkrs=1}^{M} Q_{jkrs}X_{jr}X_{ks} = (B,X) + (X,C,X)$$

$$(B,X) = \sum_{jk=1}^{M} B_{jk}X_{jk} = \sum_{jk=1}^{M} \left[\sum_{i=1}^{N} \frac{a_{ikj}{}^2}{w_i} \right] X_{jk}$$

$$(X,C,X) = \sum_{jkrs=1}^{M} X_{jk}X_{rs}C_{jkrs} = \sum_{jkrs=1}^{M} \left[-\sum_{i=1}^{N} \frac{a_{ik}a_{is}jr}{w_i{}^2} \right] X_{jk}X_{rs}$$

Note that C is negative semi-definite:

$(y,C,y) \leqslant o$ for any MxM matrix y

The gradient search was motivated by a paper by Dem'ianov (Ref. 23) Exploiting the quadratic dependence of Z and the negative-definiteness of C, one writes

$$Z(X) = Z(X^o) + (X-X^o, \text{grad } Z(X^o)) + (X-X^o,C,X-X^o)$$

where the last term is non-positive and where

$$\text{grad } Z(X^o)_{jk} = B_{jk} + 2 \sum_{rs=1}^{M} C_{jkrs} X^o{}_{rs}$$

Consequently if X is chosen to minimize $(X, \text{grad } Z(X^o))$, one finds $Z(X) \leqslant Z(X^o)$, with equality usually implying that X^o is local minimum of Z.[8] The following gradient algorithm is the result:

Step 1. Select an initial permutation matrix X^{old}.

Step 2. Compute grad $Z(X^{old})$.

Step 3. Solve $\min\limits_{X \epsilon PM} (X, \text{grad } Z(X^{old}))$ for the minimizing permutation matrix X^{new}.

Step 4. If $X^{new} \neq X^{old}$, set $X^{old} = X^{new}$ and return to Step 2; if $X^{new} = X^{old}$, stop.

The algorithm converges to a permutation matrix which generally is a local minimum of Z(X).[8] The time consuming portion of the algorithm is Step 3. The minimization in Step 3 is

---

[8] The basic property is that if $(X-X^o, \text{grad } Z(X^o)) \geq 0$ for all permutation matrices X, and if $(X-X^o, C,X-X^o) = 0$ for all X for which there is strict equality, then X^o is a local minimum for $Z(\cdot)$, where the domain of Z is now extended to the set of all doubly stochastic matrices.

$$\min_{\pi} \left[\sum_{j=1}^{M} \text{grad } Z(X^{o})_{j,\pi(j)} \right]$$

where π ranges over all permutations of $\{1,2,\ldots,M\}$.

This class of problems is known as *linear assignment problems* and is most readily solved by the so-called Hungarian method (Ref. 32). Unfortunately, the labor for the Hungarian method is proportional to M^3 or M^4, and since several linear assignment problems must be solved, the computation time for this gradient algorithm turns out to be excessive for large M.

D. COMPUTATIONAL RESULTS

The gradient algorithm described above was coded in order to provide near-optimal solutions to the Moment Compression problem. The gradient algorithm is used twice; once to minimize the sum of the row moments and again for the column moments. The major computational effort goes into solutions of successive linear assignment problems.

The primary advantages of the gradient algorithm are its simplicity and (as the following two examples illustrate) its excellent capability for putting a matrix into near block-diagonal form when this is possible. The primary disadvantage is the large computer time (a factor of three greater than for the Bond Energy Algorithm), rendering the method impractical for matrices larger than about 25x25.

The excessive computational effort arises from two sources. One is the need to solve successive linear assignment problems, each of which is time consuming. The second is the existence of several local minima for Z(X), with the consequence that the final data ordering is somewhat sensitive to the initial data ordering. (The Moment Ordering Algorithm has similar properties.) It therefore is necessary to start the algorithm at several randomly-selected initial permutations in order to achieve a final permutation for which Z is close to its global minimum. The need for multiple starts increases the computational effort many-fold.

1. First Example

A 16x16 example from Ref. 33 was solved with the gradient algorithm for moment compression. In this example, the 16 most frequently occurring non-trivial words have been extracted from a long conversation. The input matrix, A-1, is shown in Fig. 27. A "1" is placed in row i and column j if words i and j have coincidentally occurred in two or more sentences, and a "0" is placed there otherwise.

	LIPS	BED	BUS	TRACK	PERFUME	BEACH	TENNIS	HOTEL	COURT	MOUNTAIN	CLUBHOUSE	HOT	SWIM	VIEW	FIELD	LEG
LIPS	1	1	0	0	1	0	0	1	1	0	0	1	0	0	0	1
BED	1	1	0	0	1	0	0	1	1	0	0	1	0	0	0	1
BUS	0	0	1	1	0	1	0	1	0	1	0	1	0	1	0	0
TRACK	0	0	1	1	0	0	1	0	1	0	1	1	1	0	1	1
PERFUME	1	1	0	0	1	0	0	1	1	0	0	1	0	0	0	1
BEACH	0	0	1	0	0	1	0	1	0	1	1	1	0	1	0	0
TENNIS	0	0	0	1	0	0	1	0	1	0	1	0	1	0	1	1
HOTEL	1	1	1	0	1	1	0	1	1	0	0	1	0	1	0	1
COURT	1	1	0	1	1	0	1	1	1	0	1	1	1	0	1	1
MOUNTAIN	0	0	1	0	0	1	0	0	0	1	0	1	0	1	0	0
CLUBHOUSE	0	0	0	1	0	1	1	0	1	0	1	0	1	0	1	1
HOT	1	1	1	1	1	1	0	1	1	1	0	1	0	1	0	1
SWIM	0	0	0	1	0	0	1	0	1	0	1	0	1	0	1	1
VIEW	0	0	1	0	0	1	0	1	0	1	0	1	0	1	0	0
FIELD	0	0	0	1	0	0	1	0	1	0	1	0	1	0	1	1
LEG	1	1	0	1	1	0	1	1	1	0	1	1	1	0	1	1

11-13-69-5

FIGURE 27. Initial Word Relationship Array, Matrix A-1

Since the input matrix is symmetric, the problems of choosing permutations to minimize the row or column moments are identical. It sufficed to find the optimal column permutation, and to use this permutation on both the rows and columns of the matrix. The problem of row moment minimization was solved 40 times, each time starting from a randomly chosen permutation of the columns. Two solutions are taken as identical if they differ merely by reversal of the order of the 16 words.

The results were as follows. Nine of the 40 starting-points led to the final matrix, A-2, shown in Fig. 28, with the lowest ME. An additional 11 of the 40 starting-points led to a final matrix (with the same ME) which differed from A-2 only by interchanging of the variables

	MOUNTAIN	VIEW	BUS	BEACH	HOTEL	HOT	PERFUME	LIPS	BED	COURT	LEG	TRACK	CLUBHOUSE	FIELD	SWIM	TENNIS
MOUNTAIN	1	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0
VIEW	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
BUS	1	1	1	1	1	1	0	0	0	0	0	1	0	0	0	0
BEACH	1	1	1	1	1	1	0	0	0	0	0	0	1	0	0	0
HOTEL	0	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0
HOT	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0
PERFUME	0	0	0	0	1	1	1	1	1	1	1	0	0	0	0	0
LIPS	0	0	0	0	1	1	1	1	1	1	1	0	0	0	0	0
BED	0	0	0	0	1	1	1	1	1	1	1	0	0	0	0	0
COURT	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1
LEG	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1
TRACK	0	0	1	0	0	1	0	0	0	1	1	1	1	1	1	1
CLUBHOUSE	0	0	0	1	0	0	0	0	0	1	1	1	1	1	1	1
FIELD	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1
SWIM	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1
TENNIS	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1

11-13-69-7

Matrix A-2

	VIEW	BEACH	MOUNTAIN	BUS	HOT	HOTEL	LIPS	PERFUME	BED	COURT	LEG	FIELD	SWIM	CLUBHOUSE	TENNIS	TRACK
VIEW	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
BEACH	1	1	1	1	1	1	0	0	0	0	0	1	0	0	0	0
MOUNTAIN	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0
BUS	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	1
HOT	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	1
HOTEL	1	1	0	1	1	1	1	1	1	1	1	0	0	0	0	0
LIPS	0	0	0	0	1	1	1	1	1	1	1	0	0	0	0	0
PERFUME	0	0	0	0	1	1	1	1	1	1	1	0	0	0	0	0
BED	0	0	0	0	1	1	1	1	1	1	1	0	0	0	0	0
COURT	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1
LEG	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1
FIELD	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1
SWIM	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1
CLUBHOUSE	0	1	0	0	0	0	0	0	0	1	1	1	1	1	1	1
TENNIS	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1
TRACK	0	0	1	1	0	0	0	0	0	1	1	1	1	1	1	1

11-13-69-8

Matrix A-3

FIGURE 28. Reordered Word Relationship Arrays

"bed" and "lips". Since these two variables have identical rows and columns, it is understandable why ambiguity arises about their ordering.[9]

Inspection of Fig. 28 shows that the algorithm has partitioned the 16 words into three subjects: vacation, sex, and sports. The transitional words between these three topics of conversation are evidently hotel, hot, court, and leg.

All remaining 20 starting-points led to a ME which was at least 11 percent higher.[10] Inspection of the next-to-best permutation (the one with an ME 11 percent higher than that of matrix A-2) showed that the gradient algorithm converged to the wrong local minimum of Z(X), in which only one of the three topics of conversation (sports) is clearly identified.

It is believed that Matrix A-2 (and the variations obtained by permuting identical rows) achieves the global optimum of Z(X), although this is not certain. It must be recalled, however, that the primary goal is the discovery of informative patterns, not rigorous optimization of the ME. For example, the rearrangement proposed by Giuliano, Matrix A-3 in Fig. 28, is just as pleasing as A-2, even though its ME is not as good. The point here is two-fold: (1) data rearrangements with near-optimal ME may be as pleasing as those with optimal MEs; (2) ME-optimization algorithms can fail to identify *all* informative patterns, especially patterns which are not local optima for the ME.[11]

The Moment Compression Algorithm is considered to have worked properly on this example because it produced a pleasing pattern. The sensitivity of the gradient algorithm to the starting point was not a serious problem, for 20 of the 40 starts led to a good answer. Note, however, that a mere 11 percent degradation in the ME led to a seriously degraded pattern.

The main criticism of the gradient algorithm for Moment Compression is its excessive computation time. Each usage of the algorithm required 3 to 7 gradient steps (i.e., solutions of 3 to 7 linear assignment problems) at about 2 seconds per step. The algorithm therefore required about 10 seconds per starting point.[12] Since 40 starting points were chosen, at random, to ensure high confidence in achieving a global rather than local optimum,[13] 7 *minutes*[14] were required to solve this problem.

---

[9] It should be pointed out that this example exhibits considerable degeneracy. Examination of Fig. 33 reveals that rows 7-9, rows 10-11, and rows 14-16 are identical. The ME will be invariant under permutations of identical rows. In addition, rows 1-4 and rows 5-6 are nearly identical; the ME will undergo only minor changes if these are interchanged.

[10] The ME is here defined as the root-mean-square row moment,

$$ME = \sqrt{\frac{1}{16} \sum_{i=1}^{16} r_i}$$

[11] Since the starting points never led to A-3, A-3 probably is not a local minimum for Z(X).

[12] By contrast, the Bond Energy Algorithm requires only a few seconds per starting point.

[13] By contrast, the Bond Energy Algorithm is rather insensitive to the starting point, so that fewer starting points (at most 16, and probably much less) need be explored.

[14] On CDC 1604.

It may be possible to cut the running time by a factor of 2 or 4, since a "good" starting point may require fewer gradient steps than a randomized one, and by using much fewer than 40 starts. Nevertheless, the running time for a 16x16 problem (on the order of minutes, and doubled if the matrix is not symmetric) is disappointing when contrasted with the running time for the Bond Energy Algorithm. Consequently the gradient algorithm for Moment Compression Algorithm is probably impractical for problems larger than 25x25 or so. It works well, but is too slow.

2. Second Example[15]

A second example was run in order to test the ability of the gradient algorithm to generate clumps of large numbers when the matrix elements were not restricted to 0 or 1. The initial matrix is the 10x10 matrix denoted as B-1 in Fig. 29. Since B-1 is symmetric, it sufficed to find the optimal row permutation, and to use this permutation on both the rows and columns of B-1.

	A	B	C	D	E	F	G	H	I	J
A	5	4	1	0	4	1	1	0	3	1
B	4	5	0	1	3	1	1	0	4	1
C	1	0	5	0	1	3	3	0	1	2
D	0	1	0	5	0	0	0	4	0	1
E	4	3	1	0	5	1	0	0	4	1
F	1	1	3	0	1	5	3	0	1	3
G	1	1	3	0	0	3	5	0	1	2
H	0	0	0	4	0	0	0	5	1	0
I	3	4	1	0	4	1	1	1	5	1
J	1	1	2	1	1	3	2	0	1	5

11-13-69-6

FIGURE 29. Initial Similarity Matrix B-1

The gradient algorithm was used with 60 randomly chosen starting points. Four distinct MEs[16] were obtained, with values 1.846, 1.987, 1.988, and 2.314. The frequency of

---

[15] This is the same example as in Figs. 7 and 8.

[16] The ME is here defined as the root-mean square row moment-arm,

$$\mathrm{ME} = \sqrt{\frac{1}{10} \sum_{i=1}^{10} r_i}$$

these MEs were, respectively, 36, 12, 8, and 4 (sum of 60). The matrices corresponding to the four MEs (the best four permutations[17]) were respectively, B-2, B-3, B-4, and B-5 and are shown in Fig. 30. These four MEs are fairly close to one another and it is apparent that the patterns in B-2, B-3, B-4, and B-5 are essentially equivalent and equally pleasing; all four matrices succeed in identifying a 2x2 block of large numbers (variables H and D), a 4x4 block (variables A, B, E, and I) and a 4x4 block (variables C, F, G, and J). The exact order of the blocks, and of variables within each block differ, but one would be indifferent to such unimportant differences (i.e., to the arrangement of the stray 1's) since the *identification* of the blocks of primary blocks is what is significant.

The computation time for this problem was about 3 seconds per starting point. The 60 starting points consumed about 3 minutes total computation time[18].

Summarizing, the gradient algorithm applied to this problem succeeded, in all 60 of the starts, in identifying the major variable blocks and produced informative patterns. The four best permutations produced nearly equal MEs and equally informative patterns, without any obvious way of choosing among them. The algorithm is considered successful, but rather slow, for this problem. It correctly "factored" the main variables, but was very time-consuming compared with the Bond Energy Algorithm.

---

[17] Two permutations were considered equivalent if each could be obtained from the other by merely reversing the order of the 10 rows and columns.

[18] On CDC 1604 computer.

	H	D	B	I	E	A	J	F	G	C
H	5	4	0	1	0	0	0	0	0	0
D	4	5	1	0	0	0	1	0	0	0
B	0	1	5	4	3	4	1	1	1	0
I	1	0	4	5	4	3	1	1	1	1
E	0	0	3	4	5	4	1	1	0	1
A	0	0	3	4	5	4	1	1	1	1
J	0	1	1	1	1	1	5	3	2	2
F	0	0	1	1	1	1	3	5	3	3
G	0	0	1	1	0	1	2	3	5	3
C	0	0	0	1	1	1	2	3	3	5

Matrix B-2

	H	D	J	G	C	F	I	B	A	E
H	5	4	0	0	0	0	1	0	0	0
D	4	5	1	0	0	0	0	1	0	0
J	0	1	5	2	2	3	1	1	1	1
G	0	0	2	5	3	3	1	1	1	0
C	0	0	2	3	5	3	1	0	1	1
F	0	0	3	3	3	5	1	1	1	1
I	1	0	1	1	1	1	5	4	3	4
B	0	1	1	1	0	1	4	5	4	3
A	0	0	1	1	1	1	3	4	5	4
E	0	0	1	0	1	1	4	3	4	5

Matrix B-4

	H	D	G	J	C	F	I	B	A	E
H	5	4	0	0	0	0	1	0	0	0
D	4	5	0	1	0	0	0	1	0	0
G	0	0	5	2	3	3	1	1	1	0
J	0	1	2	5	2	3	1	1	1	1
C	0	0	3	2	5	3	1	0	1	1
F	0	0	3	3	3	5	1	1	1	1
I	1	0	1	1	1	1	5	4	3	4
B	0	1	1	1	0	1	4	5	4	3
A	0	0	1	1	1	1	3	4	5	4
E	0	0	0	1	1	1	4	3	4	5

11-13-69-9

Matrix B-3

	G	C	F	J	D	H	I	A	B	E
G	5	3	3	2	0	0	1	1	1	0
C	3	5	3	2	0	0	1	1	0	1
F	3	3	5	3	0	0	1	1	1	1
J	2	2	3	5	1	0	1	1	1	1
D	0	0	0	1	5	4	0	0	1	0
H	0	0	0	0	4	5	1	0	0	0
I	1	1	1	1	0	1	5	3	4	4
A	1	1	1	1	0	0	3	5	4	4
B	1	0	1	1	1	0	4	4	5	4
E	0	1	1	1	0	0	4	4	4	5

11-13-69-10

Matrix B-5

FIGURE 30. Reordered Similarity Matrices

REFERENCES

1. Ball, G.H., "Data Analysis in the Social Sciences: What About the Details?," *Proceedings - Fall Joint Computer Conference,* pp. 533-559, 1965.

2. Ball, G.H. and Hall, D.J., "ISODATA, an Iterative Method of Multivariate Analysis and Pattern Classification," presented at the Proceedings of the International Communications Conference, Philadelphia, Pa., June 1966.

3. Bonner, R.E., "On Some Clustering Techniques," *IBM Journal,* Vol. 22, pp. 22-32, 1964.

4. Parker-Rhodes, A.F. and Needham, R.H., "The Theory of Clumps," (Cambridge, England: Cambridge Language Research Unit), 1960.

5. Needham, R.M., "The Theory of Clumps II," (Cambridge, England: Cambridge Language Research Unit), Report M.L. 139, 1961.

6. Ward, J.H. and Hook, M.E., "Application of a Hierarchal Grouping Procedure to a Problem of Grouping Profiles Educational and Psychological Measurement," pp. 69-82, 1963.

7. Dale, A.G. and Dale, N., "Some Clumping Experiments for Information Retrieval: LRC 64-WPI 1," (Austin: Linguistics Research Center), February 1964.

8. Edwards, A.W.F., "A Method for Cluster Analysis," *Biometrics,* Vol. 21, pp. 372-375, 1965.

9. "Some Clumping Experiments for Associative Document Retrieval," *American Documentation,* pp. 5-9, 1965.

10. Ihm, P., "Automatic Classification in Anthropology," *Use of Computers in Anthropology,* Morton and Company, 1965.

11. Fortier, J.J. and Solomon, H., "Clustering Procedures," *Proceedings of the International Symposium on Multivariate Analysis*, pp. 493-506, New York, Academic Press, 1966.

12. King, B., "Step-Wise Clustering Procedures," *Journal of the American Statistical Association*, Vol. 62, pp. 86-101, 1967.

13. Jones, K.S., "Current Approaches to Classification and Clump Finding at Cambridge Language Research Unit," 93351, *Computer Journal*, Vol. 10, p. 29, 1967.

14. Hartigan, J., "Representation of Similarity Matrices by Trees," *Journal of American Statistical Association*, Vol. 62, pp. 1140-1158, December 1967.

15. Sokal, R.R., *Principles of Numerical Taxonomy*, W.H. Freeman and Company, San Francisco, 1963.

16. Johnson, S.C., "Hierarchical Clustering Schemes," *Psychometrika*, Vol. 63, pp. 241-254, 1967.

17. Harrison, P.J., "Method of Cluster Analysis and Some Applications," *Applied Statistics*, Vol. 17, No. 3, pp. 226-236, 1968.

18. Nagy, G., "Pattern Recognition," *Proceedings of the IEEE*, Vol. 56, p. 836, 1968.

19. MacQueen, J., "Some Methods for Classification and Analysis of Multivariate Observations," *Proceedings 5th Berkeley Symposium on Statistics and Probability* pp. 281-197, 1967.

20. Oxnard, Charles E., "The Combined Use of Multivariate and Clustering Analysis in Functional Morphology," *Journal of Biomechanics*, Vol. 2, pp. 73-88, Pergamon Press, 1969.

21. Gilmore, "Optimal and Suboptimal Algorithms for the Quadratic Assignment Problem," *Journal of the Society of Industrial and Applied Mathematics*, Vol. 10 (1962), pp. 305-313.

22. Lawler, "Quadratic Assignment Problem," *Management Science*, Vol. 9 (1962), pp. 586-599.

23. Dem'ianov and Khudiakov, "Solution of an Integer Problem of Quadratic Programming," *Applied Math. and Mechanics,* Vol. 29 No. 1 (1965), pp. 170-174.

24. Gilmore (*Ibid*).

25. Whinston and Graves, "An Algorithm for the Quadratic Assignment Model," Working Paper 110, *Western Management Science Institute,* UCLA, (revised 1968).

26. Pierce and Crowston, "Tree-Search Algorithms for Quadratic Assignment Problems," Working Paper 389-69, *Sloan School of Management,* M.I.T. (1969).

27. Hiller and Connors, "Quadratic Assignment Problem Algorithms and the Location of Indivisible Facilities," *Management Science,* Vol. 13 (1966), pp. 42-57.

28. Seal, Hilary L., *Multivariate Statistical Analysis for Biologist,* John Wiley & Sons, 1964.

29. Ahmed, R., and Agrawal, S., "Significant Features in the Perception of (Hindi) Consonants," *Journal of the Acoustical Society of America,* Vol. 45, No. 3, (1969).

30. Robinson, W.S., "A Method for Chronologically Ordering Archaeological Deposits," *American Antiquity,* Vol. 16, p. 293, 1951.

31. Bellmore, M. and Nemhauser, G.L., "The Traveling Salesman Problem: A Survey," *Operations Research,* Vol. 16, No. 3, May-June 1968.

32. James Munkres, "Algorithms for the Assignment and Transportation Problems," *Journal of the Society of Industrial and Applied Mathematics,* Vol. 5 (1957), pp. 32-38.

33. Vincent Giuliano, "How We Find Patterns," *International Science and Technology,* February 1967, pp. 40-51.

APPENDIX A

FORMULATION OF THE BOND ENERGY ME OPTIMIZATION AS TWO QUADRATIC ASSIGNMENT PROBLEMS

FORMULATION OF THE BOND ENERGY ME OPTIMIZATION AS TWO QUADRATIC ASSIGNMENT PROBLEMS

The purpose of this Appendix is to show how the problem of ME maximization can be rigorously formulated and solved as two Quadratic Assignment Problems (QAPs). This formulation is presented only for theoretical interest, because published algorithms (Refs. 21, 22, 23) which find truly optimal solutions to QAPs are too time consuming to be practical for large problems.[1] While approximate algorithms have been published[2] (Refs. 24, 25, 26, 27) which find near-optimal solutions to QAPs, it was not believed worthwhile to explore any of them, because none exploited the nearest-neighbor feature of the function being optimized. Only the sequential-selection approximate algorithm described in this paper exploits the nearest-neighbor feature, and this latter algorithm is believed to be much faster, more convenient, and just as satisfactory[3] as the published approximate QAP algorithms.

Suppose the original non-negative matrix $[a_{ij}]$ is M x N with horizontal and vertical bond energies contributing to the ME. The ME then consists of the sum of two terms, namely, the row bond energies plus the column bond energies. Two optimization problems must be solved for ME maximization. One seeks a permutation of the columns of $[a_{ij}]$ which maximizes the row bond energy, the other seeks a permutation of the rows of $[a_{ij}]$ which maximizes the column bond energy. These two optimization problems can be carried out independently of each other. When both are completed, the optimal permutations of both rows and columns are known.

The two optimization problems are mathematically equivalent. Only the problem of maximizing the row bond energy is presented here. This problem requires selection of a permutation $\pi = [\pi(1), \pi(2), \ldots, \pi(M)]$ of the integers $[1, 2, ..., M]$ which maximizes

$$\frac{1}{2} \sum_{i=1}^{N} \left\{ b_{i1} b_{i2} + \sum_{j=2}^{M-1} b_{ij} \left[b_{i,j-1} + b_{i,j+1} \right] + b_{iM} b_{i,M-1} \right\} . \qquad \text{(A-1)}$$

---

[1] Computer times on the order of one or several minutes are required for 15x15 matrices, and rise as the fourth and fifth power of the matrix size.

[2] An extensive bibliography is contained in Ref. 26.

[3] The satisfaction with the Bond Energy Algorithm is not based primarily on how close it comes to achieving the global optimum in Eq. (A-2) but rather on the pleasing patterns of clumps which it produces.

The term within braces is twice the bond energy for the i^{th} row of $[b_{ij}]$, where $[b_{ij}] = [a_{i,\pi(j)}]$ denotes the matrix whose j^{th} column is the $\pi(j)^{th}$ column of $[a_{ij}]$. The mathematical problem may be rewritten as

$$\max_{\pi} \sum_{j=1}^{M} \sum_{k=1}^{M} Q_{jk\pi(j)\pi(k)} \qquad \text{(A-2)}$$

where

$$Q_{jkrs} = \sum_{i=1}^{N} a_{ir} a_{is} [\delta_{k,j-1} + \delta_{k,j+1}] \qquad \text{(A-3)}$$

$$1 \leqslant j, k, r, s \leqslant M$$

The maximization in Eq. (A-2) is taken over all M! possible permutations. This type of maximization is known as a quadratic assignment problem because of the double appearance of π in the maximand. As previously noted, published algorithms exist for finding both optimal and near-optimal solutions to Eq. (A-2).

INTERPRETATION OF ME OPTIMIZATION AS TWO TRAVELING SALESMAN PROBLEMS

The quadratic assignment problem formulated in the previous section is actually a special type called the open-loop traveling salesman problem. Let

$$d_{rs} = \sum_{i=1}^{N} a_{ir} a_{is} = d_{sr}$$

denote the scalar product of the r^{th} and s^{th} columns of $[a_{ij}]$. Then, the maximization in (A-2) is equivalent to

$$\max_{\pi} \sum_{j=1}^{M-1} d_{\pi(j)\,\pi(j+1)} \,. \qquad \text{(A-4)}$$

If one interprets d_{rs} as the distance [4] from city r to city s, the problem in Eq. (A-4) is to find the salesman's tour [from city $\pi(1)$ to $\pi(2)$ .. to city $\pi(M)$] of the M cities which has the longest distance.[5] Note that the tour origin is arbitrary and that the salesman is not required to return to his origin. This tour is therefore called open loop.

[4] If necessary, a large positive constant can be added to all ds in order to make them positive.

[5] Subtraction of every d_{rs} from a large positive constant leads to an equivalent problem of minimizing the tour length.

APPENDIX B

PROOF THAT THE BOND ENERGY SUBOPTIMAL ALGORITHM WILL PRODUCE BLOCK FACTORED FORM IF IT IS POSSIBLE TO DO SO BY ROW AND COLUMN PERMUTATIONS

PROOF THAT THE BOND ENERGY SUBOPTIMAL ALGORITHM WILL PRODUCE BLOCK FACTORED FORM IF IT IS POSSIBLE TO DO SO BY ROW AND COLUMN PERMUTATIONS

The purpose of this appendix is to prove that the sequential selection bond energy algorithm will put a matrix into block factored form if it is possible to do so by row and column permutations.

DEFINITION 1

A non-negative matrix A whose elements a_{ij} relate row entity i to column entity j is called block factorable if the row entities can be decomposed into q disjoint subsets $R_1, R_2, \ldots R_q$, and the column entities decomposed into q disjoint subsets $C_1, C_2, \ldots C_q$ with the properties:

(1) If entity $i \in R_a$, then $a_{ij} = 0$

unless entity $j \in C_a$, $1 \leqslant a \leqslant q$

and if entity $j \in C_a$, then $a_{ij} = 0$

unless entity $i \in R_a$, $1 \leqslant a \leqslant q$

(2) For each a, the submatrix $\left\{ [a_{ij}] ,\ i \in R_a, j \in C_a \right\}$

cannot be further decomposed.

That is, A can be factored into q blocks if the row entities and column entities can each be partitioned into q subsets such that: (1) entities in one row subset interact only with entities in the corresponding column subset and (2) it is impossible to decompose the subsets further.

DEFINITION 2

A block factorable matrix is said to be in block factored form[1] when all the row entities contained in each R_a lie together on the vertical axis of the matrix and all the column entities contained in each C_a lie together on the horizontal axis of the matrix. Clearly, the matrix A is block factorable if, and only if, it can be put into block factored form by row and column permutations.

---

[1] Figure 4 shows a matrix in block factored form.

LEMMA 1

Assume A is block factorable. If row entity i of matrix A is contained in R_α and row entity j is contained in R_β with $\alpha \neq \beta$, then the scalar product of row i with row j vanishes.

Proof

For any entity k, $a_{ik}=0$ unless $k \epsilon C_\alpha$ and $a_{jk}=0$ unless $k \epsilon C_\beta$. Therefore, $a_{ik}\, a_{jk}=0$ for all k since C_α and C_β are disjoint.

LEMMA 2

Assume A is block factorable and select any R_α which contains two or more rows. No matter how R_α is split into two distinct subsets, it is always possible to choose one row from each subset such that the scalar product of the two rows is positive.

Proof

If such a choice cannot be made, then the submatrix $\{[a_{ij}],\ i \epsilon R_\alpha,\ j \epsilon C_\alpha\}$ is decomposible, violating Definition 1.

THEOREM

If A is block factorable, then the sequential selection algorithm will put the matrix into block factored form, and will do so by building one block at a time.

Proof

If the first row laid down came from (say) R_1, then the next row to be laid down will be one of the remaining M-1 rows with the greatest scalar product with the first. Since (by Lemma 1) all the rows not contained in R_1 have vanishing scalar products with the first row, and since at least one (by Lemma 2) of the as yet unplaced rows from R_1 (if any others exist) has a positive scalar product, then the second row to be laid down will come from R_1. By repeating this reasoning it is clear that all the rows from R_1 are laid down before any other rows are laid down. More generally, one subset, R_a, of row entities at a time is laid down and all the rows contained in each R_a lie together in the matrix.

Identical reasoning can be applied to show that the columns are also laid down with all the columns in each C_a lying together. Therefore, by Definition 2 the matrix will be put in block factored form.

APPENDIX C

PROOF THAT THE MOMENT COMPRESSION ALGORITHM WILL PRODUCE BLOCK FACTORED FORM IF IT IS POSSIBLE TO DO SO BY ROW AND COLUMN PERMUTATIONS

PROOF THAT THE MOMENT COMPRESSION ALGORITHM WILL PRODUCE BLOCK FACTORED FORM IF IT IS POSSIBLE TO DO SO BY ROW AND COLUMN PERMUTATIONS

The assertion here is that the minimum of the ME[1] of a matrix which can be placed in block form via row and column permutations occurs when the matrix is in block form, and does not occur when rows (or columns) of one block are separated by rows (or columns) of another block. Consequently, rigorous minimization of the ME must put the matrix into block form. Since the gradient algorithm for moment compression will find a global rather than local minimum of the ME, if sufficiently many starting points are used, it follows that the gradient algorithm will put a matrix into block form if this is possible.

It suffices to examine how column permutations can minimize $\Sigma_i r_i$. The basic idea of the proof is that if the columns from one block are separated by columns from another block, then removal of the extraneous columns, reuniting the columns from the first block, and reinsertion (at one side) of the removed columns will strictly reduce $\Sigma_i r_i$, hence reduce the ME. Thus, the ME is at its minimum only if columns from the same block are contiguous.

An example is provided by Fig. C-1 which shows the 5 left-most columns of a matrix. At least one X in each column is positive. Columns A, C, E form a block; no column to the right of E lies in this block; and columns B and D are from other blocks.

The following theorem shows that if column D is moved out to the right of the block (producing the column order A,B,C,E,D), then $\Sigma_i r_i$ will decrease. Similarly, movement of column B to the right of the block (producing the column order A,C,E,B,D) will reduce $\Sigma_i r_i$ further.

---

[1]

$$ME = \sum_{i=1}^{N} r_i + \sum_{j=1}^{M} c_j \quad \left\{ \begin{array}{l} r_i = \text{moment arm for row } i \\ c_j = \text{moment arm for column } j \end{array} \right.$$

COLUMN:	A	B	C	D	E
	X	0	X	0	X
	X	0	X	0	X
	X	0	X	0	X
	0	X	0	X	0
	0	X	0	X	0
	0	X	0	X	0
	0	X	0	X	0

FIGURE C-1. Sample Matrix

The general procedure is to identify the left-most column block whose columns are not placed contiguously,[2] and to move the right-most extraneous column[3] from the midst of the block to the immediate right of the block. Repetition of this procedure produces a column ordering which places columns from the same block in contiguous positions. Since the procedure leads to strict decreases in $\Sigma_i r_i$, it shows that the ME achieves its global ME only when the matrix is put into block factored form.

The theorem which follows shows that each r_i is decreased if the zeros which are interior to a row are moved to an edge of the block, and is unchanged if a zero at one block edge is moved to the other block edge. Thus moving columns B and D to the right of E will reduce

$$\sum_{i=1}^{3} r_i$$

because the first 3 rows have an interior (or possibly left edge) zero at column B, and an interior (or possibly right edge) zero at column D. Similarly,

$$\sum_{i=4}^{7} r_i$$

is also decreased by such a transfer because columns C and E provide interior zeros to rows 4-7.

---

[2] Initially, this is block A,C,E.

[3] Initially, this is column D.

THEOREM

$$\text{Let } W_j \geqslant 0, \ \sum_{j=1}^{N} W_j = 1, \ \bar{j} = \sum_{j=1}^{N} jW_j$$

$$S \equiv \sum_{j=1}^{N} W_j \left[j - \bar{j} \right]^2 = \text{moment for the vector W.}$$

Suppose for some k, $2 \leqslant k \leqslant N-1$, $W_k = 0$. Let * refer to a rearrangement whereby W_k has been moved to the extreme right, and the vector then closed up:

$$W_j^* = \begin{cases} W_j & 1 \leqslant j \leqslant k-1 \\ W_{j+1} & k \leqslant j \leqslant N-1 \\ 0 & j = N \end{cases} \tag{C-1}$$

$$\bar{j}^* \equiv \sum_{j=1}^{N} j W_j^* = \bar{j} - \beta \ \text{ where } \beta \equiv \sum_{j=k+1}^{N} W_j \tag{C-2}$$

$$S^* = \sum_{j=1}^{N} W_j^* \left[j - \bar{j}^* \right]^2 = \text{the moment for the vector W*.} \tag{C-3}$$

Then $S^* \leqslant S$ with equality if and only if W_k is an edge zero (that is, if $W_j=0$ for all $j \leqslant k$, or if $W_j=0$ for all $j \geqslant k$).

Proof:

Set $\bar{j} = E + F$ where

$$E = \sum_{j=1}^{k-1} w_j j \leqslant (k-1) \sum_{j=1}^{k-1} w_j = (1-\beta)(k-1) \tag{C-4}$$

$$F = \sum_{j=k+1}^{N} w_j j \geqslant (k+1) \sum_{j=k+1}^{N} w_j = \beta (k+1) . \tag{C-5}$$

Insertion of Eqs. (C-1, C-2) into Eq. (C-3) obtains

$$S^* - S = \beta(1-\beta) + 2 \sum_{j=k+1}^{N} w_j (\bar{j} - j)$$

$$= \beta(1-\beta) + 2\beta\bar{j} - 2F = \beta(1-\beta) + 2\beta E + 2(\beta - 1)F$$

Insertion of Eqs. (C-4, C-5) produces, because $0 \leqslant \beta \leqslant 1$, the result

$S^* - S \leqslant -3\beta(1-\beta) \leqslant 0$ with $S^* = S$ only when β is 0 or 1, which occurs only if w_k is an exterior zero. Thus $S^* = S$ only if w_k is an exterior zero. The converse is easily proved. QED.

Note that with the choice

$$W_j = a_{ij} \Big/ \sum_{m=1}^{N} a_{im}$$

we find $S = r_i$ = moment for i^{th} row of $[a_{ij}]$. The theorem therefore states that the * - rearrangement (namely removal of an interior zero from the i^{th} row of $[a_{ij}]$) will strictly reduce r_i unless the zero lies at an edge.

APPENDIX D

THE BOND ENERGY COMPUTER PROGRAM

THE BOND ENERGY COMPUTER PROGRAM

A. OPERATION OF THE PROGRAM

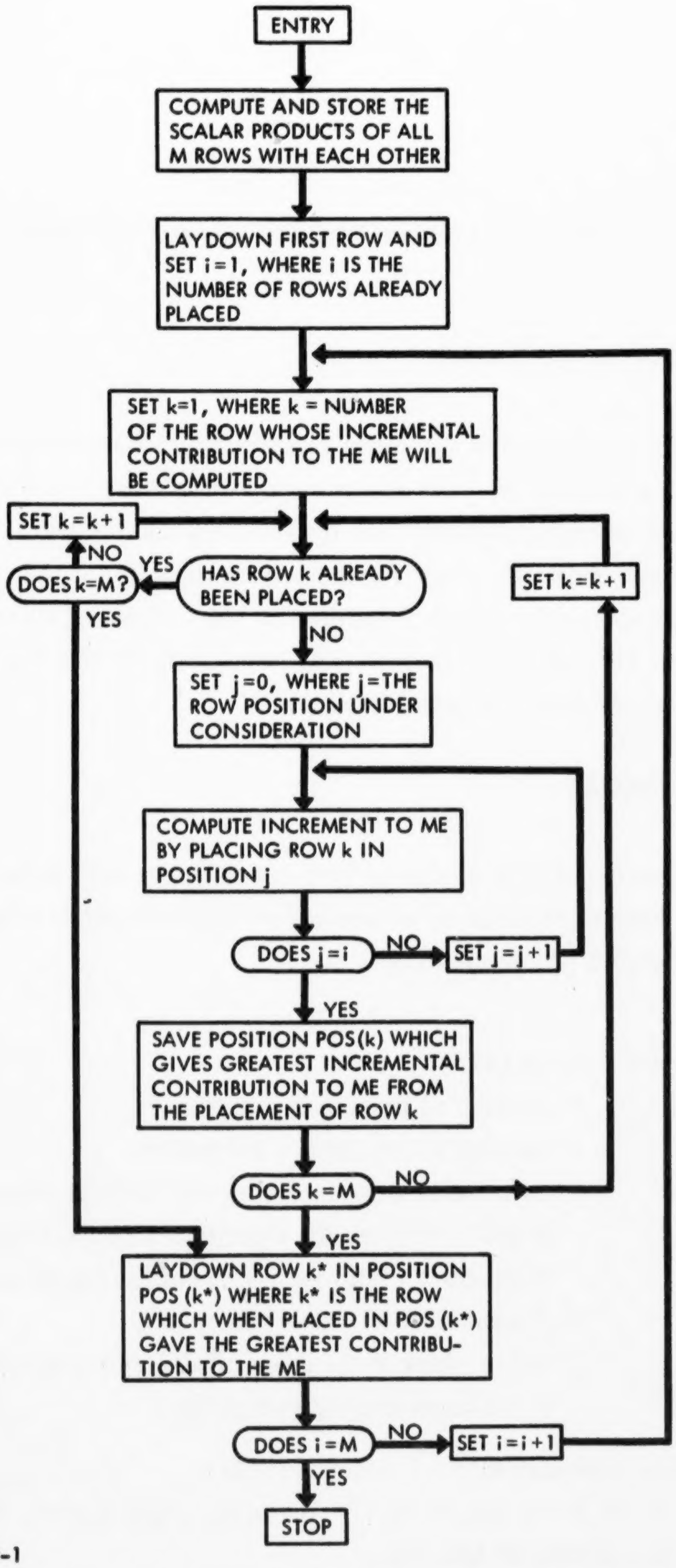
The computer program for the Bond Energy Algorithm consists of two separate parts. The first part of the program reorders the columns while the second part reorders the rows. Figure D-1 shows the essential program logic for selection and laydown of the rows to obtain a new order with a large NME. The logic for the column selection is identical. It was found that in order to examine a number of local minima it was necessary to initiate the program at several starting rows (or columns). However, as pointed out in Chapter I of Part II, almost all starting points (rows) resulted in a "good" solution.

B. CARD INPUT FORMAT

The input format that is described here is for arrays with integer elements. The only change that would have to be made to accommodate decimal entries is in the input and output formats for the initial and reordered arrays.

a. Card 1 Format (415)
MM = number of rows in the matrix
NN = number of columns in the matrix
IFZ1 = an increment to determine the starting columns or rows. If IFZ1 = 5 then the algorithm is run K times beginning with row 1 then row 6, and continuing in increments of 5 until K*IFZ1 + 1 > NN or MM
IFSYM = $\begin{cases} \text{0 or blank if the input array is not symmetric} \\ \text{1 if input array is symmetric} \end{cases}$

b. Cards 2 through MM + 1 Format (80I1)
(NA(I,J), J = 1, NN) is the I^{th} row of the input matrix. This card is repeated for all MM rows.


ENTRY
COMPUTE AND STORE THE SCALAR PRODUCTS OF ALL M ROWS WITH EACH OTHER
LAYDOWN FIRST ROW AND SET i = 1, WHERE i IS THE NUMBER OF ROWS ALREADY PLACED
SET k=1, WHERE k = NUMBER OF THE ROW WHOSE INCREMENTAL CONTRIBUTION TO THE ME WILL BE COMPUTED
SET k = k + 1
NO
YES
DOES k=M?
HAS ROW k ALREADY BEEN PLACED?
YES
SET k = k + 1
NO
SET j = 0, WHERE j = THE ROW POSITION UNDER CONSIDERATION
COMPUTE INCREMENT TO ME BY PLACING ROW k IN POSITION j
DOES j = i
NO
SET j = j + 1
YES
SAVE POSITION POS(k) WHICH GIVES GREATEST INCREMENTAL CONTRIBUTION TO ME FROM THE PLACEMENT OF ROW k
DOES k = M
NO
YES
LAYDOWN ROW k* IN POSITION POS (k*) WHERE k* IS THE ROW WHICH WHEN PLACED IN POS (k*) GAVE THE GREATEST CONTRIBUTION TO THE ME
DOES i = M
NO
SET i = i + 1
YES
STOP

10-29-69-1

FIGURE D-1. Flow Chart for Sequential Row (or Column) Selection and Laydown for Bond Energy Algorithm

C. PROGRAM OUTPUT

The computer program GROUP 2 consists of the following output information:

(1) A printout of the input cards.

(2) A printout of the new row and column orderings at each step in the sequential laydown procedure.

(3) A printout of the final reordered matrix.

(4) A printout of the final horizontal and vertical MEs.

BOND ENERGY ALGORITHM PROGRAM

```
      COMMON MM,NN
      COMMON NA(90,90),K(90),L(90),ME(90),NPOS(90),NSW
      SET INDEX G = K, H = L

      IDIM = 90
      READ 100,MM,NN,IFZ1,IFSYM
      IF(IFSYM.EQ.-0) IFSYM=0
      PRINT100,MM,NN,IFZ1,IFSYM
100   FORMAT(4I5)

      DO 1 I=1,MM
      READ 101,(NA(I,J),J=1,NN)
      DO 12 J=1,NN
      IF(NA(I,J).EQ.-0) NA(I,J)=0
 12   CONTINUE

      PRINT 102,(NA(I,J),J=1,NN)
 1    CONTINUE
 101  FORMAT(80I1)
102   FORMAT(1X,59I2)


      DO 22 KK=1,NN,IFZ1

      DO 2 I=1,MM
    2 K(I)=I
      DO 3 J=1,NN
    3 L(J) = IDIM * (J-1)
      K(1)=KK
      K(KK)=1
      ITEMP1 = L(1)
      L(1) = L(KK)
      L(KK) = ITEMP1

      NTICK=0 $ MTICK=0
      ME1=0$ME2=0
      N1=NN-1


      DO 150 NCOUNT=1,N1
      NC1=NCOUNT+1

      DO 200 J=NC1,NN
      ME(J)=0
      NPOS(J)=0

      DO 210 N=0,NCOUNT
      NSUM=1000

      DO 211 I=1,MM

      G = K(I) + L(J)
      H = K(I) + L(N)
      M = K(I) + L(N+1)

      IF(N.EQ.0)GO TO 212
      IF (NA(G).EQ.0) GO TO 213
      IF (NA(H).EQ.0) GO TO 212
      NSUM = NSUM + NA(M) * NA(G)
  212 IF(N.EQ.NCOUNT)GO TO 211
      IF (NA(M).EQ.0) GO TO 211
      NSUM = NSUM + NA(M) * NA(G)
 213  IF(N.EQ.0.OR.N.EQ.NCOUNT) GO TO 211
      IF (NA(H).EQ.0) GO TO 211
      NSUM = NSUM - NA(M) * NA(H)
  211 CONTINUE

      IF(NSUM.LT.ME(J))GO TO 210
      IF(NCOUNT.EQ.1) GO TO 214
      IF(NSUM.NE.ME(J)) GO TO 214
      IF(N.EQ.0.OR.N.EQ.NCOUNT) GO TO 214
      GO TO 210
 214  CONTINUE
      ME(J)=NSUM
      NPOS(J)=N
  210 CONTINUE

  200 CONTINUE


      MEMAX=ME(NC1)
      NPMAX=NPOS(NC1)
      IMAX=NC1
      NC2=NCOUNT+2
      DO 220 I=NC2,NN
      NSW=1
      IF(ME(I).LT.MEMAX)GO TO 220
      IF(ME(I).EQ.MEMAX) CALL EQ2I(I,IMAX)
      IF(NSW.NE.1) GO TO 220
      IMAX=I
      MEMAX=ME(I)
      NPMAX=NPOS(I)
  220 CONTINUE
      ME1=ME1+MEMAX-1000
      IF(NPMAX.NE.NCOUNT)GO TO 140
      NN1=L(IMAX)
      L(IMAX)=L(NPMAX+1)
      L(NPMAX+1)=NN1
      GO TO 150
 140  IF(NPMAX.NE.0) NTICK=NTICK+1
      NP1=NPMAX+1
      NSAVE=L(IMAX)
      NP2=NP1+1
      FOR 145 I=IMAX,NP2,1
      L(I)=L(I-1)
  145 CONTINUE
      L(NP1)=NSAVE

 150  PRINT 105,(L(I)/IDIM+1, I=1,NN)


      IF(IFSYM.EQ.0) GO TO 151
      DO 152 I=1,NN
  152 K(I)=L(I) / IDIM + 1
      ME2=ME1
      MTICK=NTICK
      GO TO 351


 151 CONTINUE
 105 FORMAT(40I3)

     M1=MM-1

     DO 350 MCOUNT=1,M1
     MC1=MCOUNT+1

     DO 400 I=MC1,MM
     ME(I)=0
     NPOS(I)=0

     DO 410 M=0,MCOUNT
     MSUM=1000

     DO 411 J=1,NN

     G = K(I) + L(J)
     H = K(M) + L(J)
     N = K(M+1) + L(J)

     IF(M.EQ.0)GO TO 412
     IF (NA(G).EQ.0) GO TO 413
     IF (NA(H).EQ.0) GO TO 412
     MSUM = MSUM + NA(H) * NA(G)
 412 IF(M.EQ.MCOUNT)GO TO 413
     IF (NA(N).EQ.0) GO TO 411
     MSUM = MSUM + NA(N) * NA(G)
413  IF(M.EQ.0.OR.M.EQ.MCOUNT) GO TO 411
     IF (NA(H).EQ.0) GO TO 411
     MSUM = MSUM - NA(H) * NA(N)
 411 CONTINUE

     IF(MSUM.LT.ME(I))GO TO 410
     IF(MCOUNT.EQ.1) GO TO 414
     IF(MSUM.NE.ME(I)) GO TO 414
     IF(M.EQ.0.OR.M.EQ.MCOUNT) GO TO 414
     GO TO 410
414  CONTINUE
     ME(I)=MSUM
     NPOS(I)=M
 410 CONTINUE

 400 CONTINUE


     MEMAX=ME(MC1)
     NPMAX=NPOS(MC1)
     JMAX=MC1
     MC2=MCOUNT+2
     DO 420 I=MC2,MM
     NSW=1
     IF(ME(I).LT.MEMAX)GO TO 420
     IF(ME(I).EQ.MEMAX) CALL EQ2J(I,JMAX)
     IF(NSW.NE.1) GO TO 420
     JMAX=I
     MEMAX=ME(I)
```

```
      NPMAX=NPOS(I)
 420  CONTINUE
      ME2=ME2+MEMAX-1000
      IF(NPMAX.NE.MCOUNT)GO TO 340
      MM1=K(JMAX)
      K(JMAX)=K(NPMAX+1)
      K(NPMAX+1)=MM1
      GO TO 350
340   IF(NPMAX.NE.0) MTICK=MTICK+1
      MP1=NPMAX+1
      MSAVE=K(JMAX)
      MP2=MP1+1
      FOR 345 J=JMAX,MP2,1
      K(J)=K(J-1)
 345  CONTINUE
      K(MP1)=MSAVE
350   PRINT 105,(K(J),J=1,MM)
 351  DO 352 G = 1,NN
 352  L(G) = L(G)/IDIM + 1
      PRINT 503, (L(J),J=1,NN)
      DO 500 I=1,MM
      PRINT 501,K(I),(NA(K(I),L(J)),J=1,NN)
 500  CONTINUE
      PRINT 502,NTICK,MTICK,ME1,ME2
22    CONTINUE
502   FORMAT(//2I5,5X,*ME1=*,I5,5X,*ME2=*,I5//)
501   FORMAT(1X,I2, 2X,57I2/)
503   FORMAT(1H1,//5X,57I2//)
      END


      SUBROUTINE EQ2I(II,IM)
      COMMON MM,NN
      COMMON NA(90,90),K(90),L(90),ME(90),NPOS(90),NSW
      SET INDEX G = K, H = L

      NSW=0
      IF(NPOS(II).EQ.NPOS(IM))GO TO 701
      RETURN

 701  DO 702 I=1,MM
      G = K(I) + L(II)
      H = K(I) + L(IM)
      IF (NA(G).NE.NA(H)) GO TO 703
 702  CONTINUE
      RETURN

 703  NSUM1=0
      NSUM2=0
      DO 704 I=1,MM
      G = K(I) + L(II)
      H = K(I) + L(IM)
      NSUM1 = NSUM1 + NA(G)
 704  NSUM2 = NSUM2 + NA(H)

      IF(NSUM1.LT.NSUM2)NSW=1
      RETURN
      END
      SUBROUTINE EQ2J(JJ,JM)
      COMMON MM,NN
      COMMON NA(90,90),K(90),L(90),ME(90),NPOS(90),NSW
      SET INDEX G = K, H = L

      NSW=0
      IF(NPOS(JJ).EQ.NPOS(JM))GO TO 701
      RETURN

  701 DO 702 I=1,NN
      G = K(JJ) + L(I)
      H = K(JM) + L(I)
      IF (NA(G).NE.NA(H)) GO TO 703
  702 CONTINUE
      RETURN


 703  NSUM1=0
      NSUM2=0
      DO 704 I=1,NN
      G = K(JJ) + L(I)
      H = K(JM) + L(I)
      NSUM1 = NSUM1 + NA(G)
  704 NSUM2 = NSUM2 + NA(H)

      IF(NSUM1.LT.NSUM2)NSW=1
      RETURN
      END


      SUBROUTINE EQ1
C     TEMPORARY SUB
      PRINT 900
 900  FORMAT(*ENTER EQ1*)
      END
      END
```

APPENDIX E

THE MOMENT ORDERING COMPUTER PROGRAM

THE MOMENT ORDERING COMPUTER PROGRAM

A. INTRODUCTION

Two versions of the program are available. One, which handles arrays of size up to 100x16, reads the arrays directly from cards. The second, which is identical to the first except in its input-output procedures, can handle arrays up to 100x100 and reads its arrays from tape.[1] Because the two are so similar, only the first will be presented here; most of the discussion, however, applies equally to both versions. A flow chart showing the main program logic is given in Fig. E-1. The following section provides instructions for the use of the program; Section C describes the program output.

B. INSTRUCTIONS FOR USE OF "MOMENT"

The following are the instructions for use of the moment program.

(1) Arrays of size up to 100x16 may be analyzed with the card input version of the program; up to 100x100 with the tape input version.

(2) Arrays for the card input version are punched onto cards, one row per card, as five-place floating-point numbers.

(3) As many separate arrays as desired may be analyzed in one computer run. Each may be solved once, or, if desired, any number of times, with the starting ordering chosen at random. In the latter case, the overall averaged solution is given as well as each individual solution.

(4) The following data cards are necessary for the program:
    (a) A card with 8 random integer digits in columns 1-8. This is always the first card of the input data deck, and is required to initialize the random number generator.

---

[1] The only modifications necessary are in the form of the input data.

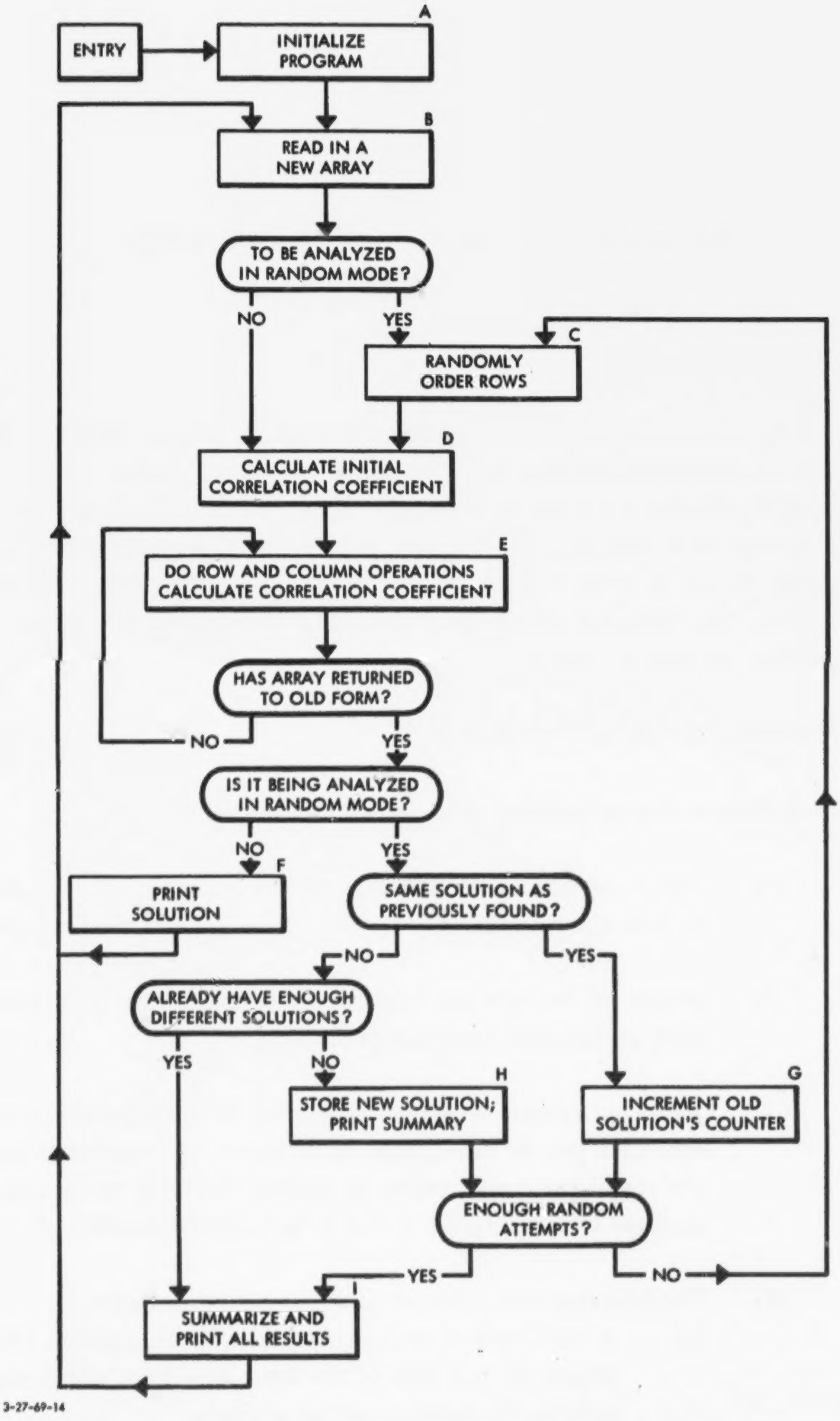

ENTRY
INITIALIZE PROGRAM
A
B
READ IN A NEW ARRAY
TO BE ANALYZED IN RANDOM MODE?
NO
YES
C
RANDOMLY ORDER ROWS
D
CALCULATE INITIAL CORRELATION COEFFICIENT
E
DO ROW AND COLUMN OPERATIONS CALCULATE CORRELATION COEFFICIENT
HAS ARRAY RETURNED TO OLD FORM?
NO
YES
IS IT BEING ANALYZED IN RANDOM MODE?
NO
YES
F
PRINT SOLUTION
SAME SOLUTION AS PREVIOUSLY FOUND?
NO
YES
ALREADY HAVE ENOUGH DIFFERENT SOLUTIONS?
YES
NO
H
STORE NEW SOLUTION; PRINT SUMMARY
G
INCREMENT OLD SOLUTION'S COUNTER
ENOUGH RANDOM ATTEMPTS?
YES
NO
I
SUMMARIZE AND PRINT ALL RESULTS
3-27-69-14

FIGURE E-1. Flow Chart for Moment Ordering Algorithm

(b) A card with "FINIS" in columns 1-5 as the *last* card of the data deck.

(c) In between the previous two cards, separate packets of data cards, one for each array. If an array is to be analyzed in the random mode, the first card of that packet must contain the number of random tries (an integer) to be carried out in columns 1-4, and the work "RANDOM" in columns 9-14. If this option is not to be exercised, this card is merely omitted. The next card (therefore the first card for the one-time-only option) contains the number of rows (integer) in columns 1-4, the number of columns (integer) in columns 5-8, and the name of the array in columns 9-80. This is followed by the cards containing the array proper.

(d) As many packets of cards for individual arrays as desired may follow each other. A sample data deck for the card input version is shown in Fig. E-2. (A deck for the tape input version could be identical except that it would not have the cards with the arrays themselves punched.)

```
COL. 1
↓
82340795                           ←RANDOM NUMBER
   3   5SAMPLE DATA                ←HEADER CARD, ARRAY 1
1.0  3.0  0.0  0.54  11.3          }
8.4  2.7  6.11 12.04 18.1          } ARRAY 1
0.0  14.0 111. 6.08  1.0           }
  10    RANDOM                     ←RANDOM CARD, ARRAY 2
   4   2MORE SAMPLE DATA           ←HEADER CARD, ARRAY 2
1.0  2.4                           }
1.0  3.7                           } ARRAY 2
0.0  2.15                          }
1.0  6.0                           }
   2   3MORE SAMPLE DATA           ←HEADER CARD, ARRAY 3
4.1  7.9  6.1                      } ARRAY 3
4.2  0.0  7.0                      }
FINIS                              ←LAST CARD
11-13-69-13
```

FIGURE E-2. Sample Data Deck for Card Input Version of Moment Ordering Algorithm

C. "MOMENT" PROGRAM OUTPUT

The program output consists of the following:

(1) For each array analyzed, the program prints out a complete copy of the array as read in, numbering the rows and columns.

(2) If only a single analysis is to be done, the program first prints out the initial value of the correlation coefficient, R, and then after each row or column operation prints out the entire array, numbering the rows and columns appropriately and giving the new correlation coefficient. When it reaches a solution, it prints the array and the message "THIS IS THE SOLUTION."

(3) If the random-ordering, multiple-attempt option is being employed, each time a new solution is found the program prints out the order of the rows and the value of the correlation coefficient. When it has finished the appropriate number of attempts, it prints out a complete copy of the most-commonly-found solution and lists all of the solutions found, giving the correlation coefficient, the number of times found, and the order of the rows for each. It then lists the overall averaged solution, giving the ordering of the rows and columns, and the average position (with the RMS deviation) of each row and column.

(4) If the program encounters an unstable array structure (i.e., one in which no solution may be found, but rather a cycle of states occurs which will repeat itself indefinitely), it takes that as a solution, but first prints the message "THE FOLLOWING SOLUTION IS NOT STABLE."

MOMENT ORDERING ALGORITHM PROGRAM

```
      PROGRAM MOMENT
      TRACE ARRAYS
C     STEPHEN DEUTSCH,EXT.358,SED
      COMMON A(100,16)
      DIMENSION NARHAY(5000)
      DIMENSION NSOLS(25,16)
      DIMENSION MSOLS(25,100)
      DIMENSION          M(100),N(16),MT(100),NT(16),MTO(100),NTO(16)
      DIMENSION DM(100),DMO(100),DN(16),DNO(16),IEXT(9)
      DIMENSION ROLD(100),NOLD(100)
      DIMENSION RSAVE(100)
C     I AND M REFER TO COLUMNS, J AND N TO ROWS
      EPS=1.E-6
      TLEFT=0.
      TNOW=CLOCKTM(TLEFT)
      IF(TLEFT.LT.1000.) GO TO 14
      CALL SLURP(NARRAY,5000,0)
   14 CONTINUE
      READ 12,NDUM
   12 FORMAT(I8)
      RAN=RAN1F(NDUM)
    1 CALL READIN(MC,MSIZE,NSIZE,IEXT,M,N,NMC,NUM,NMAX)
   20 IF(MC.EQ.1) CALL RANORD(M,MSIZE,NMC)
      R0=-2.
      NLIST=0
      CALL CC(MSIZE,NSIZE,  M,N,R)
      IF(MC.NE.1)PRINT 108,R
  100 CONTINUE
      CALL NID(  M,N,MSIZE,NSIZE,R,MTO,NTO,DN,DNO,MT,NT,IEXT,MC)
      CALL DUMP(5)
      IF(MC.NE.1)PRINT 108,R
      CALL MID(  M,N,MSIZE,NSIZE,R,MTO,NTO,DM,DMO,MT,NT,IEXT,MC)
      CALL DUMP(5)
      IF(MC.NE.1)PRINT 108,R
  108 FORMAT(//* R=*,F10.5)
      IF(NLIST.EQ.0) GO TO 9
      DO 11 I=1,NLIST
      IF(ABS(R-RSAVE(I)).GE.EPS) GO TO 11
      IF(I.NE.NLIST) PRINT 112
  112 FORMAT(* THE FOLLOWING SOLUTION IS NOT STABLE*)
      GO TO 110
   11 CONTINUE
    9 NLIST=NLIST+1
      RSAVE(NLIST)=R
      IF(NLIST.EQ.100) NLIST=0
      GO TO 100
  110 CONTINUE
      IF(MC.EQ.1)GO TO 2
      CALL PRINTA(MSIZE,NSIZE,  M,N,MT,NT,IEXT)
      PRINT 109
  109 FORMAT(//* THIS IS THE SOLUTION*)
      GO TO 1
    2 IF(NUM.EQ.0) GO TO 10
      DO 6 I=1,NUM
      IF(ABS(R-ROLD(I)).LT.EPS) GO TO 7
    6 CONTINUE
   10 IF(NUM.GE.25) GO TO 8
      NUM=NUM+1
      ROLD(NUM)=R
   NOLD(NUM)=1
15 DO 18 I=1,MSIZE
18 MSOLS(NUM,I)=M(I)
   DO 13 I=1,NSIZE
13 NSOLS(NUM,I)=N(I)
 5 PRINT 3,(MT(I),I=1,MSIZE)
 3 FORMAT(/1X,35I3/1X,35I3/1X,30I3)
   PRINT 4,R
 4 FORMAT(4H R= ,F10.5)
21 IF(NMC.GE.NMAX) GO TO 8
   GO TO 20
 7 NOLD(I)=NOLD(I)+1
   GO TO 21
 8 CALL SUM(IEXT,NUM,MSIZE,NSIZE,DMO,DNO,MSOLS,NSOLS,ROLD.NOLD,NMC)
   GO TO 1
   END

   SUBROUTINE READIN(MC,MSIZE.NSIZE,IEXT,M,N,NMC,NUM,NMAX)
   COMMON A(100,16)
   DIMENSION           M(100),N(16),MT(100),NT(16),IEXT(9)
   DATA(IRAN=6HRANDOM)
   MC=0
 3 READ 2,MSIZE,NSIZE,IEXT
 2 FORMAT(2I4,9A8)
   NMC=0
   NUM=0
   IF(IRAN.EQ.IEXT(1)) GO TO 1
   DO 4 I=1,MSIZE
   READ 5,(A(I,J),J=1,NSIZE)
 4 CONTINUE
 5 FORMAT(16F5.2)
   DO 6 I=1,NSIZE
 6 N(I)=I
   DO 7 I=1,MSIZE
 7 M(I)=I
   CALL PRINTA(MSIZE,NSIZE,  M7N,MT,NT,IEXT)
   PRINT 11,IEXT
11 FORMAT(//10H1SOLUTIONS/1X,9A8/)
   RETURN
 1 NMAX=MSIZE
   MC=1
   GO TO 3
   END

   SUBROUTINE RANORD(M,MSIZE,NMC)
   COMMON A(100,16)
   DIMENSION M(100)
   DIMENSION RANS(100)
 9 DO 10 I=1,MSIZE
   RANS(I)=RAN1F(-1)
10 CONTINUE
   CALL SORT2(RANS,M,MSIZE,1)
   NMC=NMC+1
   RETURN
   END

   SUBROUTINE SUM(IEXT,NUM,MSIZE,NSIZE,DMO,DNO,MSOLS,NSOLS,ROLD,
  1NOLD,NMC)
   COMMON A(100,16)
   DIMENSION MX(100)
```

```
      DIMENSION MSOLS(25,100)
      DIMENSION NSOLS(25,16)
      DIMENSION AVE(100),DEV(100),NORDER(100)
      DIMENSION ROLD(100),NOLD(100)
      DIMENSION IEXT(9),MT(100),NT(16),DMO(MSIZE),DNO(NSIZE)
      DIMENSION NX(16)
   12 ISOL=1
      IF(NUM.LT.2) GO TO 26
      DO 22 I=2,NUM
   22 IF(NOLD(I).GT.NOLD(ISOL)) ISOL=I
   26 MZ=0
      DO 33 I=1,MSIZE
   33 MX(I)=MSOLS(ISOL,I)
      DO 34 I=1,NSIZE
   34 NX(I)=NSOLS(ISOL,I)
      CALL PRINTA(MSIZE,NSIZE,MX,NX,MT,NT,IEXT)
      PRINT 35
   35 FORMAT(///* THIS IS THE MOST COMMON SOLUTION*)
      PRINT 13,IEXT
   13 FORMAT(1H1/* SUMMARY*/1X,9A8//8X,*R*,6X,*TIMES FOUND    ROW ORDER*/)
     1/)
      DO 25 I=1,MSIZE
   25 IF(DMO(I).EQ.0.) MZ=MZ+1
      MLO=MZ+(MSIZE-MZ)/3
      MHI=MSIZE+1-(MSIZE-MZ)/3
      MG=MZ+1
      NMZ=0
      DO 14 I=1,NSIZE
   14 IF(DNO(I).EQ.0.) NMZ=NMZ+1
      DO 23 II=1,NUM
      DO 24 I=1,MSIZE
      MX(I)=MSOLS(II,I)
   24 MT(I)=I
      CALL SORT2(MX,MT,MSIZE,-1)
      MOK=0
      DO 27 I=MG,MLO
      MVAL=MSOLS(ISOL ,MT(I))
      IF(MVAL.LE.MLO) MOK=MOK+1
   27 IF(MVAL.GE.MHI) MOK=MOK-1
      DO 28 I=MHI,MSIZE
      MVAL=MSOLS(ISOL,MT(I))
      IF(MVAL.LE.MLO) MOK=MOK-1
   28 IF(MVAL.GE.MHI) MOK=MOK+1
      IF(MOK.GT.0) GO TO 31
      DO 29 I=1,MSIZE
   29 IF(MSOLS(II,I).GT.MZ) MSOLS(II,I)=MSIZE+MZ+1-MSOLS(II,I)
      DO 37 I=1,NSIZE
   37 IF(NSOLS(II,I).GT.NMZ) NSOLS(II;I)=NSIZE+NMZ+1-NSOLS(II,I)
      DO 30 I=1,MSIZE
      MX(I)=MSOLS(II,I)
   30 MT(I)=I
      CALL SORT2(MX,MT,MSIZE,-1)
   31 PRINT 32,ROLD(II),NOLD(II),(MT(J),J=1,MSIZE)
   32 FORMAT(1X,F10.5,I8,5X,32I3/18X,34I3/18X,34I3)
   23 CONTINUE
      PRINT 15
   15 FORMAT(///* AVERAGED SOLUTION*///*  ROW    AVERAGE      DEVIATION*
     1/)
      AZ=NMC
      DO 16 I=1,MSIZE
      AVE(I)=0.
      DO 16 J=1,NUM
      AX=MSOLS(J,I)
      AY=NOLD(J)
      AVE(I)=AVE(I)+AX*AY/AZ
   16 NORDER(I)=I
      DO 17 I=1,MSIZE
      DEV(I)=0.
      DO 18 J=1,NUM
      AX=MSOLS(J,I)
      AY=NOLD(J)
   18 DEV(I)=DEV(I)+AY*(AX-AVE(I))**2/AZ
   17 DEV(I)=SQRT(DEV(I))
      CALL SORT2(AVE,NORDER,MSIZE,-1)
      DO 19 I=1,MSIZE
      PRINT 20,NORDER(I),AVE(I),DEV(NORDER(I))
   20 FORMAT(I5,2F10.2)
   19 CONTINUE
      PRINT 38
   38 FORMAT(///* AVERAGED SOLUTION*///*  COL.    AVERAGE     DEVIATION*
     1/)
      DO 36 I=1,NSIZE
      AVE(I)=0.
      DO 36 J=1,NUM
      AX=NSOLS(J,I)
      AY=NOLD(J)
      AVE(I)=AVE(I)+AX*AY/AZ
   36 NORDER(I)=I
      DO 41 I=1,NSIZE
      DEV(I)=0.
      DO 39 J=1,NUM
      AX=NSOLS(J,I)
      AY=NOLD(J)
   39 DEV(I)=DEV(I)+AY*(AX-AVE(I))**2/AZ
   41 DEV(I)=SQRT(DEV(I))
      CALL SORT2(AVE,NORDER,NSIZE,-1)
      DO 40 I=1,NSIZE
      PRINT 20,NORDER(I),AVE(I),DEV(NORDER(I))
   40 CONTINUE
      PRINT 21
   21 FORMAT(1H1)
      END

      SUBROUTINE PRINTA(MSIZE,NSIZE,  M,N,MT,NT,IEXT)
      COMMON A(100,16)
      DIMENSION MX(100)
      DIMENSION            M(MSIZE),N(NSIZE),MT(MSIZE),NT(NSIZE)
      DIMENSION IEXT(9)
      DO 3 I=1,MSIZE
      MX(I)=M(I)
    3 MT(I)=I
      CALL SORT2(MX,MT,MSIZE,-1)
      DO 5 I=1,NSIZE
      MX(I)=N(I)
    5 NT(I)=I
      CALL SORT2(MX,NT,NSIZE,-1)
      PRINT 10,IEXT,NT
   10 FORMAT(1H1,9A8///1H ,6X,16I7/)
      DO 100 I=1,MSIZE
      PRINT 200,MT(I),(A(MT(I),NT(J)),J=1,NSIZE)
```

```
100 CONTINUE
200 FORMAT(1H0,I6,16F7.3)
    END

    SUBROUTINE M1D(   M,N,MSIZE,NSIZE,R,MTO,NTO,DM,DMO,MT,NT,IEXT,MC)
    COMMON A(100,16)
    DIMENSION MX(100)
    DIMENSION             M(MSIZE),N(NSIZE),MT(MSIZE),NT(NSIZE)
    DIMENSION MTO(MSIZE),NTO(NSIZE),DM(MSIZE),DMO(MSIZE)
    DIMENSION ITABLE(16),ITIE(100),MTSAVE(100),IEXT(9)
    EPS=1.E-5
    DO 3 I=1,MSIZE
    MX(I)=M(I)
  3 MT(I)=I
    CALL SORT2(MX,MT,MSIZE,-1)
    DO 5 I=1,NSIZE
    MX(I)=N(I)
  5 NT(I)=I
    CALL SORT2(MX,NT,NSIZE,-1)
    DO110 I=1,MSIZE
110 MTO(I)=MT(I)
    DO115 J=1,NSIZE
115 NTO(J)=NT(J)
    DO 20 I=1,MSIZE
    D1=0.
    X1=0.
    DO 15 J=1,NSIZE
    D1=D1+A(MT(I),NT(J))
    XN=J
    X1=X1+XN*A(MT(I),NT(J))
 15 CONTINUE
    DM(I)=X1/D1
 20 CONTINUE
    DO 12 I=1,MSIZE
 12 DMO(MT(I))=DM(I)
    CALL ORDER(DM,MSIZE,MT,M,MTO)
    RBEST=-2.0
    KK=MSIZE-1
    DO 21 I=1,KK
    ITIE(I)=0
    J=I+1
    IF(DMO(MT(I)).EQ.DMO(MT(J)))ITIE(I)=1
    IF(DMO(MT(I)).EQ.0.) ITIE(I)=0
 21 CONTINUE
    ITIE(MSIZE)=0
    NTIED=1
    I=0
 22 I=I+1
    IF(ITIE(I).NE.0)GO TO 23
    IF(NTIED.NE.1)GO TO 24
    IF(I.GE.KK)GO TO 25
    GO TO 22
 23 NTIED=NTIED+1
    GO TO 22
 24 ITABLE(1)=-1
    IF(NTIED.GT.9) PRINT 34
    IF(NTIED.GT.9) NTIED=9
 34 FORMAT(*MORE THAN  9 ROWS OR COLUMNS TIED WHILE DOING ALGORITHM, F
   1IRST 9 IN TIE WERE REORDERED, OTHERS LEFT IN OLD ORDER*)
    DO 27 II=1,MSIZE
```

```
27 MTO(II)=MT(II)
   RBEST=-2.0
   NFACT=1
   DO 28 II=1,NTIED
28 NFACT=NFACT*II
   DO 29 II=1,NFACT
   CALL PERMUTE(NTIED,ITABLE)
   DO 30 JJ=1,NTIED
30 MTO(I-NTIED+JJ)=MT(I-NTIED+ITABLE(JJ))
   DO 31 JJ=1,MSIZE
   MX(JJ)=MTO(JJ)
31 M(JJ)=JJ
   CALL SORT2(MX,M,MSIZE,-1)
   CALL CC(MSIZE,NSIZE,  M,N,R)
   IF(R.LT.(RBEST-EPS))GO TO 29
   RBEST=R
   DO 32 JJ=1,MSIZE
32 MTSAVE(JJ)=MTO(JJ)
29 CONTINUE
   DO 33 JJ=1,MSIZE
   MT(JJ)=MTSAVE(JJ)
   MX(JJ)=MTSAVE(JJ)
33 M(JJ)=JJ
   CALL SORT2(MX,M,MSIZE,-1)
   NTIED=1
   IF(I.LT.KK)GO TO 22
25 R=RBEST
   IF(R.EQ.-2.0)CALL CC(MSIZE,NSIZE,  M,N,R)
   IF(MC.EQ.1)RETURN
   PRINT 10,IEXT,NT
10 FORMAT(1H1,9A8///1H ,2X,16I7/)
   DO 100 I=1,MSIZE
   PRINT 200,MT(I),(A(MT(I),NT(J)),J=1,NSIZE),DMO(MT(I))
100 CONTINUE
200 FORMAT(1H0,I2,16F7.3,F5.2)
   RETURN
   END

   SUBROUTINE N1D(  M,N,MSIZE,NSIZE,R,MTO,NTO,DN,DNO,MT,NT,IEXT,MC)
   COMMON A(100,16)
   DIMENSION MX(100)
   DIMENSION             M(MSIZE),N(NSIZE),MT(MSIZE),NT(NSIZE)
   DIMENSION MTO(MSIZE),NTO(NSIZE),DN(NSIZE),DNO(NSIZE)
   DIMENSION ITABLE(16),ITIE(100),MTSAVE(100),IEXT(9)
   EPS=1.E-5
   DO 3 I=1,MSIZE
   MX(I)=M(I)
 3 MT(I)=I
   CALL SORT2(MX,MT,MSIZE,-1)
   DO 5 I=1,NSIZE
   MX(I)=N(I)
 5 NT(I)=I
   CALL SORT2(MX,NT,NSIZE,-1)
   DO110 I=1,MSIZE
110 MTO(I)=MT(I)
   DO115 J=1,NSIZE
115 NTO(J)=NT(J)
   DO 20 J=1,NSIZE
   D1=0.
   X1=0.
```

```
      DO 15 I=1,MSIZE
      D1=D1+A(MT(I),NT(J))
      XN=I
      X1=X1+XN*A(MT(I),NT(J))
   15 CONTINUE
      DN(J)=X1/D1
   20 CONTINUE
      DO 12 I=1,NSIZE
   12 DNO(NT(I))=DN(I)
      CALL ORDER(DN,NSIZE,NT,N,NTQ)
      RBEST=-2.0
      KK=NSIZE-1
      DO 21 I=1,KK
      ITIE(I)=0
      J=I+1
      IF(DNO(NT(I)).EQ.DNO(NT(J)))ITIE(I)=1
      IF(DNO(NT(I)).EQ.0.) ITIE(I)=0
   21 CONTINUE
      ITIE(NSIZE)=0
      NTIED=1
      I=0
   22 I=I+1
      IF(ITIE(I).NE.0)GO TO 23
      IF(NTIED.NE.1)GO TO 24
      IF(I.GE.KK)GO TO 25
      GO TO 22
   23 NTIED=NTIED+1
      GO TO 22
   24 ITABLE(1)=-1
      IF(NTIED.GT.9) PRINT 34
      IF(NTIED.GT.9) NTIED=9
   34 FORMAT(*MORE THAN  9 ROWS OR COLUMNS TIED WHILE DOING ALGORITHM, F
     1IRST 9 IN TIE WERE REORDERED, OTHERS LEFT IN OLD ORDER*)
      DO 27 II=1,NSIZE
   27 NTO(II)=NT(II)
      RBEST=-2.0
      NFACT=1
      DO 28 II=1,NTIED
   28 NFACT=NFACT*II
      DO 29 II=1,NFACT
      CALL PERMUTE(NTIED,ITABLE)
      DO 30 JJ=1,NTIED
   30 NTO(I-NTIED+JJ)=NT(I-NTIED+ITABLE(JJ))
      DO 31 JJ=1,NSIZE
      MX(JJ)=NTO(JJ)
   31 N(JJ)=JJ
      CALL SORT2(MX,N,NSIZE,-1)
      CALL CC(MSIZE,NSIZE,  M,N,R)
      IF(R.LT.(RBEST-EPS))GO TO 29
      RBEST=R
      DO 32 JJ=1,NSIZE
   32 NTSAVE(JJ)=NTO(JJ)
   29 CONTINUE
      DO 33 JJ=1,NSIZE
      NT(JJ)=NTSAVE(JJ)
      MX(JJ)=NTSAVE(JJ)
   33 N(JJ)=JJ
      CALL SORT2(MX,N,NSIZE,-1)
      NTIED=1
      IF(I.LT.KK)GO TO 22
```

```
 25 R=RBEST
    IF(R.EQ.-2.0)CALL CC(MSIZE,NSIZE,  M,N,R)
    IF(MC.EQ.1)RETURN
    PRINT 10,TEXT,NT
 10 FORMAT(1H1,9A8///1H ,2X,16I7/)
    DO 100 I=1,MSIZE
    PRINT 200,MT(I),(A(MT(I),NT(J))7J=1,NSIZE)
100 CONTINUE
200 FORMAT(1H0,I2,16F7.3)
    II=0
    PRINT 200,II,(DNO(NT(J)),J=1,NSIZE)
    RETURN
    END

    SUBROUTINE ORDER(D,MSIZE,MT,M,MTO)
    COMMON A(100,16)
    DIMENSION MX(100)
    DIMENSION D(MSIZE),MT(MSIZE),M(MSIZE),MTO(MSIZE)
    CALL SORT2(D,MTO,MSIZE,-1)
    DO 60 I=1,MSIZE
    MT(I)=MTO(I)
 60 M(I)=I
    CALL SORT2(MTO,M,MSIZE,-1)
    END

    SUBROUTINE CC(MSIZE,NSIZE,  M,N;R)
    COMMON A(100,16)
    DIMENSION            M(MSIZE),N(NSIZE)
    DIMENSION XLIST(100),YLIST(100)
    SX=0.
    SY=0.
    SSX=0.
    SSY=0.
    SPXY=0.
    SP=0.
    XLM=MSIZE
    XLN=NSIZE
    XLM=1./XLM
    XLN=1./XLN
    DO 1 J=1,NSIZE
    X=N(J)
  1 YLIST(J)=X*XLN
    DO 2 I=1,MSIZE
    X=M(I)
  2 XLIST(I)=X*XLM
    DO 100 I=1,MSIZE
    Z=XLIST(I)
    DO 100 J=1,NSIZE
    ZZ=YLIST(J)
    P=A(I,J)
    XP=P*Z
    YP=P*ZZ
    SX=SX+XP
    SY=SY+YP
    SSX=SSX+XP*Z
    SSY=SSY+YP*ZZ
    SPXY=SPXY+XP*ZZ
100 SP=SP+P
    SSDX=SSX-(SX)**2/SP
    SSDY=SSY-(SY)**2/SP
```

```
SPDXY=SPXY-SX*SY/SP
RR=SQRT(SSDX*SSDY)
IF(RR.EQ.0.) R=0,0
IF(RR.NE.0.) R=SPDXY/RR
ENDSEND%
```

APPENDIX F

THE MOMENT COMPRESSION COMPUTER PROGRAM

THE MOMENT COMPRESSION COMPUTER PROGRAM

A. INTRODUCTION

This Appendix describes the computer program MOMCOMP which implements the gradient algorithm in Chapter III of Part II. The main program logic is shown in the flow chart in Fig. F-1.

B. DATA RESTRICTIONS

The program can handle an indefinite number of non-negative matrices, each up to 75x75 in size. The data packages for successive matrices are stacked one after another.

C. FORMAT FOR INPUT DATA

The listing of MOMCOMP includes, for illustrative purposes, the data package required as input for the second example in Chapter III of Part II. This is a 10x10 matrix and three starting points are requested for the row or column optimizations.

The data input package for each matrix consists of two types of cards. The first card type contains the three variables N, M, IRAN punched according to format (3I5).

Here

N = number of matrix rows

M = number of matrix columns

IRAN = number of randomized starts for the optimization in each direction.(Row and column optimizations therefore consume 2 · IRAN starts.)

The second type of data card is used to read in a single row of the input matrix. Successive rows are punched on distinct cards, with each card employing format (60I1).

If the matrix contains up to 60 columns exactly, N+1 data cards will be required, namely one of type 1 and N of type 2.

D. SUBROUTINE DESCRIPTION

The portions of MOMCOMP include:

(1) Main Program, which controls execution.

(2) DATA subroutine, which reads input data.

(3) PERMGEN subroutine, which generates randomized permutations for use as starting points.

(4) ZMIN subroutine, which minimizes the sum of either the row or column moments, using the iterative gradient algorithm described in Chapter III of Part II.

(5) LAP subroutine, used by ZMIN, which solves linear assignment problems.

(6) Z function, which computes the objective function, taken as

$$Z = \begin{cases} \sqrt{\frac{1}{N} \sum_{i=1}^{N} r_i} & \text{for optimization of column order} \\ \sqrt{\frac{1}{M} \sum_{j=1}^{M} c_j} & \text{for optimization of row order} \end{cases}$$

Z is the root-mean-square moment arm.

(7) Subroutines INITCOL and INITROW, which prepare the data needed by ZMIN for minimizing Z. The required data are the W's and H's in the expression (see Eq. 8-9) in Chapter III of Part II).

$$Z(\pi) = \sqrt{\frac{1}{L}\left[\sum_{i=1}^{L} W_j \pi_j - \sum_{ij=1}^{L} H_{ij}\pi_i\pi_j\right]}$$

where L = N(M) for optimization of the column (row) order, and where $\pi_1, \pi_2, \ldots, \pi_L$ denotes a permutation of the L columns (rows).

E. PROGRAM FLOW CHART

Figure F-1 is the computer program flow chart for the Moment Compression Algorithm.

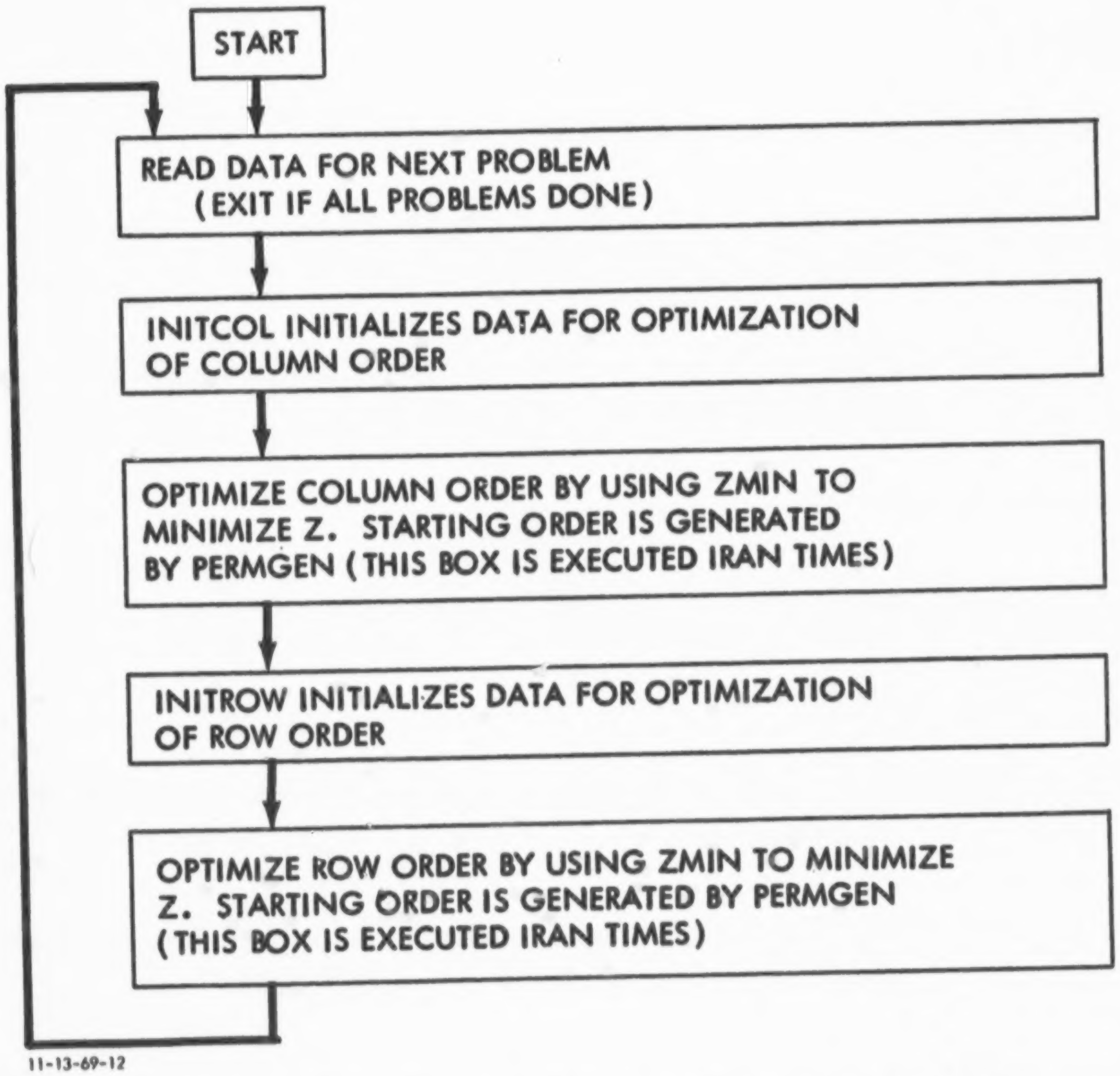

START
READ DATA FOR NEXT PROBLEM (EXIT IF ALL PROBLEMS DONE)
INITCOL INITIALIZES DATA FOR OPTIMIZATION OF COLUMN ORDER
OPTIMIZE COLUMN ORDER BY USING ZMIN TO MINIMIZE Z. STARTING ORDER IS GENERATED BY PERMGEN (THIS BOX IS EXECUTED IRAN TIMES)
INITROW INITIALIZES DATA FOR OPTIMIZATION OF ROW ORDER
OPTIMIZE ROW ORDER BY USING ZMIN TO MINIMIZE Z. STARTING ORDER IS GENERATED BY PERMGEN (THIS BOX IS EXECUTED IRAN TIMES)
11-13-69-12

FIGURE F-1. Flow Chart for Moment Compression Algorithm

F. ERROR MESSAGES

(1) The only error message from ZMIN is that convergence has not occurred after 100 iterations (each iteration is one gradient step and involves one linear assignment problem).

(2) Five possible ERROR messages may occur in LAP:
Type 1: Step 1 has been unsuccessful at covering all zeros
Type 2: Same
Type 3: Matrix element for the linear assignment problem is negative (should never occur)
Type 4: Step 2 fails to find a primed zero in the indicated row (should never occur)
Type 5: Same as Type 3 error (should never occur).

G. TIMING

The following times should be multiplied by twice IRAN.

(1) 10x10 problem consumed roughly 3 seconds for optimization of row (or column) order for each starting point.

(2) 16x16 problem consumed roughly 10 seconds for optimization of row (or column) order for each starting point.

H. COMPUTER PRINTOUT

The computer printout from MOMCOMP consists of the following:

(1) The input data

(2) The number of the starting point (ranging from 1 through IRAN) and whether the row or column order is being optimized

(3) For each starting point, the sequence of permutations and Zs generated by successive iterations of the subroutine ZMIN.

There is no attempt to choose among the several solutions obtained by varying the starting points and no printout of "the" final matrix since this is generally non-unique. The user must extract, from the printout, the row and column permutations which minimize their respective Zs.

The computer output must be interpreted as follows. If the row permutation is printed as $\pi(1), \ldots, \pi(N)$, then the optimal rearrangement has, as its $\pi(i)^{th}$ row, the i^{th} row of the input matrix. The user is reminded that the permutations $[\pi(1), \ldots, \pi(N)]$ and $[N+1-\pi(1), \ldots, N+1-\pi(N)]$ are equivalent, each being the other in reversed order.

MOMENT COMPRESSION ALGORITHM PROGRAM

```
10   10   3
      PROGRAM MOMCOMP
C     P,SCHWEITZER X314
      COMMON W(100),H(75,75),D(75,75),C(75,75),IRAN,N,M
      DIMENSION IPERM(100)
C     INITIALIZE RANDOM NUMBER GENERATOR
      X= RANDN(3.1416)
C     FINDS IRAN LOCAL MIN FOR ROW AND COLUMN PERMUTATIONS.
1     CONTINUE
      CALL TIME(39,17HBEGIN NEW MATRIX.)
      CALL DATA
C     READS DATA
      CALL TIME(39,11HDATA IS IN. )
      CALL INITCOL
      CALL TIME(39,13HINITCOL DONE.)
C     COMPUTES W AND H FOR COLUMN PERMUTATIONS.
      DO 20 I=1,IRAN
      L=M
      CALL PAGESKP
      PRINT 50,I
      CALL PERMGEN(IPERM,L)
C     CALL TIME(39,13HPERMGEN DONE.)
C     GENERATES PERMUTATION OF 1-M,STORES IN IPERM.
      CALL TIME(39,41HZMIN STARTS OPTIMIZATION OF COLUMN ORDER.)
      CALL ZMIN(L,IPERM)
C     ZMIN FINDS OPTIMAL COLUMN PERMUTATION, STARTING FROM IPERM.
      CALL TIME(39,39HZMIN ENDS OPTIMIZATION OF COLUMN ORDER.)
 20   CONTINUE
      CALL INITROW
      CALL TIME(39,13HINITROW DONE.)
C     COMPUTES W AND H FOR ROW PERMUTATION
      DO 30 I=1,IRAN
      L=N
      CALL PAGESKP
      PRINT 50,I
 50   FORMAT(*PERGMEN CALLED FOR TIME*,I5)
      CALL PERMGEN(IPERM,L)
      CALL TIME(39,13HPERMGEN DONE.)
      CALL TIME(39,38HZMIN STARTS OPTIMIZATION OF ROW ORDER.)
      CALL ZMIN(L,IPERM)
      CALL TIME(39,36HZMIN ENDS OPTIMIZATION OF ROW ORDER.)
 30   CONTINUE
```

```
C       FINDS OPTIMAL ROW PERMUTATION, STARTING FROM IPERM,
        GO TO 1
        END
C
C
C
C
        SUBROUTINE DATA
        COMMON W(100),H(75,75),D(75,75),C(75,75),IRAN,N,M
        CALL PAGESKP
        READ 10,N,M,IRAN
10      FORMAT(3I5)
        PRINT 15,N,M,IRAN
15      FORMAT(*NEW CASE. N,M,IRAN =  *,3I5)
C       DO 18 I=1,N
C18     READ 20,(D(I,J),J=1,M)
20      FORMAT(7F10.5)
        DIMENSION ID(100)
        DO 25 I=1,N
        READ 21,(ID(J),J=1,M)
21      FORMAT(60I1)
        DO 23 J=1,M
23      D(I,J)= ID(J)
25      CONTINUE
        PRINT 30,(((I,J,D(I,J)),J=1,M),I=1,N)
30      FORMAT(/*ORIGINAL MATRIX*//*I   J   D(I,J)  */(2I5,F10.3))
        PRINT 40
40      FORMAT(* DATA ALL READ IN *)
        RETURN
        END
C
C
C
        SUBROUTINE INITCOL
        COMMON W(100),H(75,75),D(75,75),C(75,75),IRAN,N,M
        DIMENSION S(100)
        DO 5 I=1,N
        S(I)=0.
        DO 3 J=1,M
3       S(I) =S(I)+ D(I,J)
5       CONTINUE
        DO 15 J=1,M
        W(J)=0.
        DO 10 I=1,N
10      W(J)= W(J)+D(I,J)/S(I)
15      CONTINUE
        DO 30 J=1,M
        DO 19 K=1,J
        X=0.
        DO 17 I=1,M
17      X=X+D(I,J)*D(I,K)/S(I)**2
19      H(J,K)=X
30      CONTINUE
        DO 40  J=1,M
        DO 35 K=J,M
35      H(J,K)= H(K,J)
40      CONTINUE
C       CALL PAGESKP
C       PRINT 50,((J,W(J)),J=1,M)
C50     FORMAT(/*INITCOL DONE.    J W(J) */(I5,E15.8))
```

```
C       PHINT 60,(((J,K,H(J,K)),K=1/M),J=1,M)
C60     FORMAT(/*J  K  H(J,K)  */(2I5,E16.8))
        RETURN
        END
C
C
C
C
        SUBROUTINE INITROW
        COMMON W(100),H(75,75),D(75/75),C(75,75),IRAN,N,M
        DIMENSION S(100)
        DO 5 I=1,M
        S(I)=0.
        DO 3 J=1,N
3       S(I) =S(I)+ D(J,I)
5       CONTINUE
        DO 15 J=1,N
        W(J)=0.
        DO 10 I=1,M
10      W(J)= W(J)+D(J,I)/S(I)
15      CONTINUE
        DO 30 J=1,N
        DO 19 K=1,J
        X=0.
        DO 17 I=1,N
17      X=X+D(J,I)*D(K,I)/S(I)**2
19      H(J,K)=X
30      CONTINUE
        DO 40  J=1,N
        DO 35 K=J,N
35      H(J,K)= H(K,J)
40      CONTINUE
C       CALL PAGESKP
C       PRINT 50,((J,W(J)),J=1,N)
C50     FORMAT(/*INITHOW DONE.    J W(J) */(I5,E15.8))
C       PRINT 60,(((J,K,H(J,K)),K=1/N),J=1,N)
C60     FORMAT(/*J  K  H(J,K)  */(2I5,E16.8))
        RETURN
        END
C
C
C
        SUBROUTINE PEHMGEN(IPERM,N)
        DIMENSTON IPERM(100),XX(100)
        DO 10 I=1,N
        IPERM(I)=I
10      XX(I)=RANDN(0)
        CALL SORT2(XX,IPERM,N,1)
        RETURN
        END
C
C
C
        FUNCTION Z(M,IPERM)
        COMMON W(100),H(75,75),D(75/75),C(75,75),IRAN,NROW,MCOL
        DIMENSTON IPERM(100)
        Z=0.
        DO 10 J=1,M
        Z=Z+W(J)*FLOATF(IPERM(J))**2
        DO 5 K=1,M
```

```
5       Z=Z-H(J,K)*FLOATF(IPERM(J)*IPERM(K))
10      CONTINUE
        Z= SQRTF(Z/FLOATF(M))
        RETURN
        END
C
C
C
        SUBROUTINE ZMIN(M,IPERM)
        COMMON W(100),H(75,75),D(75/75);C(75,75),IRAN,NROW,MCOL
        DIMENSIONIPERM(100),ICURPER(100),NEWPER(100),Y(100)
        ZNEW= Z(M,IPERM)
        PRINT 10,  ((I,IPERM(I)),I=1,M)
10      FORMAT(/* ZMIN INITIALIZED AS FOLLOWS, PERMUTATION*/(2I5))
        PRINT 15,ZNEW
15      FORMAT(*OBJECTIVE FCN= *,E20.8)
        DO 20 I=1,M
20      NEWPER(I)= IPERM(I)
        DO 100 LLL=1,100
C       CALL TIME(39,2AHZMIN BEGINS   ONE ITERATION,)
        LL=LLL
C       DO UP TO 100 LOOPS OF ITERATIONS
        DO 21 I=1,M
21      ICURPER(I)= NEWPER(I)
        ZOLD= ZNEW
C       GENERATE C MATRIX
        DO 25  J=1,M
        Y(J)= 0.
        DO 23 K=1,M
23      Y(J)= Y(J)-2.*H(J,K)*FLOATF(ICURPER(K))
25      CONTINUE
        DO 30 J=1,M
        DO 28 K=1,M
28      C(J,K)= W(J)*FLOATF(K)**2+ Y(J)*FLOATF(K)
30      CONTINUE
        IL= LAP(  M,NEWPER,OBJ)
        IF (IL.EQ .1)  RETURN
        ZNEW= Z(M,NEWPER)
        PRINT 40,LL,ZOLD,ZNEW
40      FORMAT(* ZMIN ITERATION NUMBER *,I3,*,OLD AND NEW OBJECTIVE FUNCTI
       1ONS= *,2E20.8)
        PRINT 45,((I,NEWPER(I)),I=17M)
45      FORMAT(*NEW PERMUTATION*/(2I5))
        IF(ZNEW-ZOLD) 100,60,150
60      DO 70 I=1,M
        IF(NEWPER(I).NE.ICURPER(I)) GO TO 100
70      CONTINUE
        GO TO 150
100     CONTINUE
150     PRINT 160,LL
160     FORMAT(*ITERATIONS END AT LOOP*;I5)
        RETURN
        END
C
C
C
        FUNCTION LAP(  N,IPERMOT,OBJ)
        COMMON W(100),H(75,75),D(75775);C(75,75),IRAN,NROW,MCOL
C       LINEAR ASSIGNMENT PROBLEM SOLVER WITH N X N C.   C IS DESTROYED.
C       RETURN WITH LAP=0 IF SUCCESSFUL7 1 IF UNSUCCESSFUL.
```

```
C     OUTPUT IS OPTIMAL PERMUTATION IN IPERMOT, MINIMUM OBJECTIVE FCN IN OBJ
      DIMENSION                      IPERMOT(100),IZERO(500),JZERO(500),
     1MARK(500), MCOLCOV(100),MROWCOV(100),LIST(200)
      DIMENSION U(100),V(100)
C     CALL TIME(39,10HLAP BEGINS)
      SUB=0.
3     ISTEP1=0
      IERROR=0
C     SUB= CUMULATIVE AMOUNT SUBTRACTED FROM A ROW OR COLUMN.
C     PRINT 1,(((I,J,C(I,J)),J=1,N), I=1,N)
C1    FORMAT(2I5,E18.7,2I5,E18.7,2I5, E18.7,2I5,E18.7 )
C     SUBTRACT SMALL ELEMENT FROM EACH ROW
      DO 10 I=1,N
      X=C(I,1)
      DO 5 J=2,N
5     X= AMIN1 (X,C(I,J))
      SUB= SUB+X
      DO 10 J=1,N
10    C(I,J)= C(I,J)-X
C     SUBTRACT SMALL EST ELEMENT FROM EACH COLUMN
      DO 20 J=1,N
      X=C(1,J)
      DO 15 I=2,N
15    X= AMIN1 (Y,C(I,J))
      SUB= SUB+X
      DO 20  I=1,N
20    C(I,J)= C(I,J)-X
C     PRINT 1,(((I,J,C(I,J)),J=1,N),I=1,N)
C     STORE ALL ZEROS IN A ONE-DIMENSIONAL ARRAY. NZERO ZEROS.
C     K-TH ZERO IS AT IZERO(K),JZERO(K).
C     MARK(K)= 1 IF THIS ZERO IS STARRED, =-1 IF IT IS PRIMED, =0 OTHERWISE
      NZERO=0
      DO 30 I=1,N
      DO 30  J=1,N
      IF(C(I,J))  25,27,30
25    C(I,J) =0.
      PRINT 26,I,J
26    FORMAT(/*ELEMENT*,2I5,*RESET TO ZERO*)
      GO TO 600
27    NZERO= NZERO+1
      IZERO(NZERO)= I
      JZERO(NZERO)=J
      MARK(NZERO)=0
30    CONTINUE
35    FORMAT(  *ZERO PRINTOUT*/(20I5))
C     INITIALIZE COVERS
C     MCOLCOV(I)=1 IF COLUMN I IS COVERED,0 IF UNCOVERED, SIM. FOR MROWCOV.
C     INITIALIZE STARS
      NSTAR=0
C     NSTAR= NUMBER OF STARRED ENTRIES
C     DO INITIAL STARRING BY SUCCESSIVELY CHOOSING ZEROS WHICH HAVE THE LEAST
C     NUMBER OF OTHER ZEROS IN THE COVERERED PORTION OF EITHER ROW OR COLUMN.
C     THEN COVER THAT ROW OR COLUMN.
      DO 2000 I=1,N
      MROWCOV(I)=0
2000  MCOLCOV(I)=0
2100  ICOVER=0
      MINLINE=2*N
      MINSUM=4*N
C     AT END OF PASS, ICOVER=0 MEANS ALL ZEROS ARE COVERED. ICOVER POSITIVE
```

```
C      MEANS UNCOVERED ZEROS EXIST; AND LBEST IS THE BEST OF THEM, HAVING THE
C      FEWEST(MINLINE) OTHER UNCOVERED ZEROS IN ITS ROW OR COLUMN.
       DO 2200 K=1,NZERO
C      COMPUTE  NUMBER OF UNCOVERED ZEROS IN LINE WITH K-TH,EXCLUDING K ITSELF.
       IF(MROWCOV(IZERO(K)).EQ.1) GO TO 2200
       IF(MCOLCOV(JZERO(K)).EQ.1) GO TO 2200
C      K IS UNCOVERED.
       ICOVER=1
       INKROW=0
       INKCOL=0
C      INKROW(INKCOL)=NO. OF UNCOVERED ZEROS INROW(COLUMN) WITH K
       DO 2150 L=1,NZERO
       IF(IZERO(K).EQ.IZERO(L)) GO TO 2120
       IF(JZERO(K).NE.JZERO(L)) GO TO 2150
C      L IS IN SAME COLUMN AS K AND L.NE.K.  NOW TEST IFUNCOVERED.
       IF(MROWCOV(IZERO(L)).EQ.1) GO TO 2150
       INKCOL=INKCOL+1
       GO TO 2150
2120   IF(MCOLCOV(JZERO(L)).EQ.1) GO TO 2150
       IF(L.EQ.K) GO TO 2150
C      L IS IN SAME ROW AS K, DISTINCT; AND UNCOVERED.
       INKROW=INKROW+1
2150   CONTINUE
       INKLINE= MIN0(INKROW,INKCOL)
       INKSUM=INKROW+INKCOL
       IF(INKLINE.GT.MINLINE) GO TO 2200
       IF(INKLINE.EQ.MINLINE.AND.INKSUM.GE.MINSUM) GO TO 2200
       MINLINE= INKLINE
       MINSUM= INKSUM
       LBEST= K
       IF(MINLINE.EQ.0.AND.MINSUM.LE.1) GO TO 2300
2200   CONTINUE
       IF(ICOVER.EQ.0) GO TO 2500
C      UNCOVERED ZERO AT LBEST IS NOW COVERED.
2300   MROWCOV(IZERO(LBEST))=1
       MCOLCOV(JZERO(LBEST))=1
       MARK(LBEST)=1
       NSTAR=NSTAR+1
C      PRINT 2450, IZERO(LBEST),JZERO(LBEST)
C2450 FORMAT(*INITIALIZATION COVERS ZERO AT*, 2I5)
       GO TO 2100
2500   CONTINUE
C      STARRING OF ZEROS DONE.COLUMNS PROPERLY COVERED; UNCOVER ROWS
       DO 40 I=1,N
40     MROWCOV(I)=0
C      PRINT 85,((K,IZERO(K),JZERO(K),MARK(K)),K=1,NZERO)
C      PRINT 80,NSTAR
C80    FORMAT( *PRELIMINARIES DONE; *  ,I5,*STARS,*     )
       IF(NSTAR.EQ.N) GO TO 400
C
C
C
C  STEP 1
100    ISTEP1= ISTEP1+1
C      CALL TIME(39,16HLAP BEGINS STEP1)
       IF(ISTEP1.GT.500) GO TO 603
       ISTEPFL=0
101    IFLAG=0
       ISTEPFL = ISTEPFL+1
       IF(ISTEPFL.GT.2*NZERO) GO TO 604
```

```
C     AT END OF LOOP,AT 180, IFLAG =0 IF NO UNCOVERED ZEROS EXIST, AND
C     =1 IF POSSIBLY UNCOVERED ZEROS EXIST. EXTRA PASSES  THROUGH THIS LOOP
C     APPEAR TO BE NECESSARY TO ENSURE NO UNCOVERED ZEROS EXIST.
      DO 180 K=1,NZERO
      KK=K
      IK= IZERO(K)
      JK= JZERO(K)
      IF((MROWCOV(IK)+MCOLCOV(JK)).GT;0) GO TO 180
      IFLAG=1
      MARK(K)=-1
C     PRINT 125,IK,JK
C125  FORMAT(   *STEP1 HAS PRIMED AN UNCOVERED ZERO AT*,2I5)
C     IF THERE NO STARRED ZERO IN ROW IK, GO TO STEP 2. IF THERE IS A ZERO
C     AT L, COVER THIS ROW AND UNBOVER THE COLUMN OF L.
C     REPEAT TILL ALL ZEROS ARE COVERED, THEN GO TO STEP 3.
      DO 130 L=1,NZERO
      IF(IZERO(L).NE.IK) GO TO 130
      IF(L.EQ.K) GO TO 130
      IF(MARK(L).NE.1) GO TO 130
      GO TO 150
130   CONTINUE
C     PRINT 135
C135  FORMAT(*STEP 1 FOUND NO STARRED ZERO IN THIS ROW,WENT TO STEP 2*)
      GO TO 200
150   I=IZERO(L)
      J=JZERO(L)
C     PRINT 155,I,J
C155  FORMAT(*STEP 1 FOUND STARRED ZERO IN ROW, AT *, 2I5)
      MROWCOV(I)=1
      MCOLCOV(J)=0
      IFLAG=1
180   CONTINUE
      IF(IFLAG) 190,190,195
190   CONTINUE
C190  PRINT 191,ISTEPFL
C191  FORMAT(  *STEP 1 DONE IN*,I5,*  PASSES, ON TO STEP 3,*)
      GO TO 300
195   CONTINUE
C195  PRINT 196,ISTEPFL
C196  FORMAT(*STEP1 UNDONE AFTER*,I5, *  PASSES, RESUME,*)
      GO TO 101
C
C
C
C
C     STEP 2. MY KP1= 1+MUNKRES K
C     LIST(J) CONTAINS THE NUMBER OF ZSUB(J-1)
C     AT 210, STEP2 ASSUMES ZSUB(KP1) EXISTS AND SEARCHES FOR ZSUB(KP1+1)
200   KP1=1
C     CALL TIMF(39,16HLAP BEGINS STEP2)
      LIST(1)= KK
210   JTEST= JZERO(LIST(KP1))
      DO 220 L=1,NZERO
      IF(MARK(L).NE.1) GO TO 220
      IF(JZERO(L).EQ.JTEST) GO TO 260
220   CONTINUE
C     SEQUENCE HAS TERMINATED.
C     PRINT 221,((I,LIST(I),IZERO(LIST(I)),JZERO(LIST(I))),I=1,KP1)
C221  FORMAT(//*ZERO SEQUENCE FOR STEP 2*/(4I5))
C STAR PRIMES IN SEQUENCE, UNSTAR STARS IN SEQUENCE
```

```
      DO 230 NK=1,KP1
      IF(MARK(LIST(NK))) 222,230,225
222   MARK(LIST(NK))=1
      GO TO 230
225   MARK(LIST(NK))=0
230   CONTINUE
C     ERASE ALL PRIMES, UNCOVER ALL ROWS, COVER EVERY COLUMN CONTAINING
C     A STARRED ZERO.
      DO240 I=1,N
      MCOLCOV(I)=0
240   MROWCOV(I)=0
      NSTAR=0
      DO 250  K=1,NZERO
      IF(MARK(K))242,250,245
242   MARK(K)=0
      GO TO 250
245   MCOLCOV(JZERO(K))=1
      NSTAR= NSTAR+1
250   CONTINUE
C     PRINT 255,NSTAR
C255  FORMAT(  *STEP 2 HAS TERMINATED WITH *,I5, *STARS*)
C     PRINT 35,((K,IZERO(K),JZERO(K),MARK(K)),K=1,NZERO)
C     PRINT 325,((I,MROWCOV(I),MCOLCOV(I)),I=1,N)
      IF(NSTAR.EQ.N) GO TO 400
      GO TO 100
260   KP1=KP1+1
      LIST(KP1)=L
      ITEST= IZERO(L)
C     A PRIMED ZERO IN ROW ITEST IS GUARANTEED TO EXIST.
      DO 270  K=1,NZERO
      IF(MARK(K).NE.-1)  GO TO 270
      IF(IZERO(K).EQ.ITEST) GO TO 280
270   CONTINUE
      GO TO 601
280   KP1= KP1+1
      LIST(KP1)= K
      GO TO 210
C
C
C
C     STEP 3
300   IF(NSTAR.EQ.N) GO TO 400
C     CALL TIME(39,16HLAP BEGINS STEP3)
C     LESS THAN N INDEPENDENT STARRED ZEROS
C     COMPUTE X, THE MINIMUM UNCOVERED NUMBER. IT IS STRICTLY POSITIVE.
C     ADD X TO EACH COVERED ROW, SUBTRACT X FROM EACH UNCOVERED COLUMN.
C     PRINT 320,NZERO
C320  FORMAT(*STEP 3 BEGINS WITH*, I5; *ZEROS, *)
C     PRINT 325,((I,MROWCOV(I),MCOLCOV(I));I=1,N)
C325  FORMAT(*COVER PRINTOUT*/(3I5))
      X= 1.0E+100
      DO 351 I=1,N
      IF(MROWCOV(I).EQ. 1) GO TO 351
      DO 350 J=1,N
      IF(MCOLCOV(J).EQ.1) GO TO 350
      X=AMIN1 (X,C(I,J))
350   CONTINUE
351   CONTINUE
C     PRINT 360,X
C360  FORMAT(/*STEP 3 SUBTRACTS*, E20:8)
```

```
      IF(X.LF.0.) PRINT 370
370   FORMAT(//*STEP 3 ERROR. MINIMUM UNCOVERED ELEMENT NON-POSITIVE*)
      IF (X.LE.0.) GO TO 602
      DO 380 I=1,N
      IF(MROWCOV(I).EQ.0) GO TO 380
      SUB= SUB-X
      DO 375 J=1,N
375   C(I,J)= C(I,J)+X
380   CONTINUE
      DO 390 J=1,N
      IF(MCOLCOV(J).EQ.1) GO TO   390
      SUB= SUB+X
      DO 385 I=1,N
385   C(I,J) = C(I,J)-X
390   CONTINUE
C     DETETE THE ZEROS WHICH BECAME POSITIVE. THESE ARE PRECISELY THE
C     TWICE-COVERED ZEROS
      K=0
3900  K=K+1
3901  IF(K.GT.NZERO) GO TO 3920
      IF((MROWCOV(IZERO(K))+MCOLCOV(JZERO(K))), NE.2) GO TO 3900
C     PRINT 3905,IZERO(K),JZERO(K)
C3905 FORMAT(* STEP 3 DELETES ZERO AT *,  2I5)
C     DELETE K-TH ZERO, PUTTING LAST ZERO IN THIS SLOT.
      IF(K.EQ.NZERO) GO TO  3910
      MARK(K)= MARK(NZERO)
      MARK(NZERO)=0
      IZERO(K)= IZERO(NZERO)
      IZERO(NZERO)=0
      JZERO(K)= JZERO(NZERO)
      JZERO(NZERO)=0
      NZERO=NZERO-1
      GO TO 3901
3910  MARK(NZERO)=0
      IZERO(NZERO)=0
      JZERO(NZERO)=0
      NZERO= NZERO-1
3920  CONTINUE
C     ADD ANY NEW ZEROS TO LIST. THESE CAN ONLY BE IN THE UNCOVERED  AREA,
C     SINCE ALL ZEROS ON LIST ARE COVERED.
      DO 395 I=1,N
      IF(MROWCOV(I).EQ,1) GO TO 395
      DO 394  J=1,N
      IF(MCOLCOV(J).EQ,1) GO TO 394
      IF(C(I,J)) 392,393,394
392   C(I,J)=0.
      PRINT 26,I,J
393   NZERO=NZERO+1
      IZERO(NZERO)=I
      JZERO(NZERO)= J
      MARK(NZERO)=0
394   CONTINUE
395   CONTINUE
C     PRINT 396 ,NZERO
C396  FORMAT(  *STEP 3 DONE. *,I5/ *ZEROS IN MATRIX.*)
C     PRINT 1,(((I,J,C(I,J)),J=1,N),I=1,N)
C     PRINT 35,((K,IZERO(K),JZERO(K),MARK(K)),K=1,NZERO)
C     PRINT 325,((I,MROWCOV(I),MCOLCOV(I)),I=1,N)
      GO TO 100
C
```

```
C
C
C
C     DONE , N STARS EXIST.
400   OBJ= SUB
      DO 410 K=1,N
410   IPERMOT(K)=0
      DO 420  K=1,NZERO
      IF(MARK(K).NE.1) GO TO 420
      IPERMOT(IZERO(K))=JZERO(K)
420   CONTINUE
C     PRINT 430,OBJ
C430  FORMAT(*LAP SUCCESSFUL. OBJECTIVE = *,E20.8//* OPTIMAL PLACE  FOR
C    1I  *)
C     PRINT 440,((I,IPERMOT(I)),I=1,N)
C440  FORMAT(2I5)
C     PRINT 1,(((I,J,C(I,J)),J=1,N),I=1,N)
      LAP=0
      RETURN
C
C
C     ERROR MESSAGES
600   IERROR= IERROR+1
601   IERROR= IERROR+1
602   IERROR= IERROR+1
603   IERROR= IERROR+1
604   IERROR= IERROR+1
      PRINT 620,IERROR
620   FORMAT(*LAP ERROR OF TYPE*  ,I5)
      LAP=1
      RETURN
      END
      END
```

APPENDIX G

DISCUSSION OF MEASURES OF EFFECTIVENESS

DISCUSSION OF MEASURES OF EFFECTIVENESS

A. RELATIONSHIP BETWEEN PLEASING PATTERNS AND MEASURES OF EFFECTIVENESS

The ultimate goal of any data-organizing algorithm is the discovery of an informative pattern of variable relationships, as evidenced by a pleasing matrix appearance.

Quantitative measures of effectiveness are used as *surrogates* for pleasing patterns, since the latter concept is an intuitive one not easily described in words. The two MEs used in this report, the summed bond energies and the summed moments of inertia, were chosen with the hope that they would produce pleasing patterns by creating dense blocks of numbers. No doubt other MEs can be devised for this purpose; the two proposed here are useful because they are both (1) amenable to simple algorithms for approximate optimization and (2) successful at producing informative patterns. Any other useful ME must share these two properties.

The algorithms used for optimization of these two MEs (the sequential selection algorithm for the bond energy ME, and the gradient algorithm for the moment of inertia ME) are suboptimal, that is, they do not rigorously optimize their respective MEs. Neither algorithm should be faulted for producing suboptimal solutions, because the ultimate goal is producing informative patterns, *not* rigorously optimizing the ME; the ME is merely a surrogate for measuring the pleasingness of a pattern. Indeed, the satisfaction with the two algorithms is based upon their producing data orderings which are informative.

It often happens that several appealing data arrangements exist, all with approximately the same ME (namely, near the optimum), and all very similar.

Consequently, ties or near-ties among the ME can only be broken by a subjective eyeball judgment as to which data arrangement is most pleasing. Until the eyeball judgment is made, the tying and near-tying configurations must be considered equally aceptable. For example, the five solutions in Table 2, or the four solution matrices in Figure 30, and the two solutions in Figure 28 must be considered equally acceptable. It is highly arbitrary to choose one over the others on the basis of the numerical value of the ME.

In short, the ME is useful only for the first-order task of locating a handful of good arrangements. The ME is not useful, except in an arbitrary way, for the second-order task of choosing among the good (and nearly equally pleasing) arrangements.

B. GENERALIZATION OF THE BOND ENERGY ME

The bond energy ME can be generalized to include bonds between matrix elements which are not nearest neighbors. For example, a ME which weights the bonds according to the inverse square of the distance between the matrix elements (so-called gravity model) would be

$$\mathrm{ME} = \sum_{ij} \sum_{rs \neq ij} \frac{a_{ij}\, a_{rs}}{(i-r)^2 + (j-s)^2}$$

It may be conjectured that such generalized MEs, when optimized over all row and column permutations, are more successful at producing tightly clumped matrix elements than the ME used for the Bond Energy Algorithm, which involved only nearest neighbor bonds. There are two objections to the generalized ME, however. One is the significantly greater computational difficulty in optimizing the ME over all row and column permutations. Once the nearest neighbor feature is abandoned, the sequential selection procedure described in Appendix D cannot be used.[1] Even more serious is the fact that when diagonal bonds are included in the ME, it is no longer possible to optimize the ME in two passes, one which optimizes the row order, the other optimizing the column order. Instead, one would probably have to iterate, as in the moment ordering algorithm, between row rearrangements and column rearrangements.

The second objection to a generalized ME which includes diagonal bonds is that, for sparse matrices, optimization of the ME may result in numerous bonds being attached to the large matrix elements, thereby actually *destroying* the pleasing pattern. An example of this phenomenon is given by the case

$$d = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 500 \end{bmatrix} \qquad d^1 = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 500 & 0 \\ 1 & 0 & 1 \end{bmatrix}$$

Note that row and column permutations can transform d^1 into d. Since d is in block form, it conveys more information about the group structure and is preferable to d^1. However, if any of the three following MEs (which allow next-to-nearest neighbor diagonal bonds) are used, then d^1 is preferable to d. Evidently optimization of the bond energies leads to the attachment of as many (diagonal) bonds to the "500" as possible, even at the expense of block form. By contrast, when the diagonal bonds are excluded, optimization of the ME will produce block form if this is possible. (See Appendix B.)

---

[1] This procedure can be modified, however, if the generalized ME includes only row-bonds and column-bonds (albeit not necessarily nearest neighbor), and lacks diagonal bonds.

The three MEs are

$$ME_1(b) = \sum_{ij} \delta_{ij}\, \alpha_{ij}$$

$$ME_2(b) = \sum_{ij} a_{ij}\, \alpha_{ij}$$

$$ME_3(b) = \sum_{ij} a_{ij}\, \beta_{ij}$$

$$\text{where } \delta_{ij} = \begin{cases} 1 & b_{ij} > 0 \\ 0 & b_{ij} = 0 \end{cases}$$

$$\alpha_{ij} = a_{i,j+1} + a_{i,j-1} + a_{i+1,j} + a_{i-1,j} + 1/2\left[a_{i+1,j+1} + a_{i+1,j-1} + a_{i-1,j+1} + a_{i-1,j-1}\right]$$

$$\beta_{ij} = \delta_{i,j+1} + \delta_{i,j-1} + \delta_{i+1,j} + \delta_{i-1,j} + 1/2\left[\delta_{i+1,j+1} + \delta_{i+1,j-1} + \delta_{i-1,j+1} + \delta_{i-1,j-1}\right]$$

If any of the three are used, then d^1 has a higher ME than d.

$ME_1(d) = 510$ $\quad ME_1(d^1) = 1002$
$ME_2(d) = 510$ $\quad ME_2(d^1) = 2000$
$ME_3(d) = 260.5$ $\quad ME_3(d^1) = 1002$

C. ADDITIONAL PROPERTIES OF THE MEs

(1) All three algorithms are unaffected if all the matrix elements are multiplied by a positive constant.

(2) All three algorithms are affected if a constant is added to the matrix element. In particular this implies sensitivity to the choice of origin of the ordinal scale (e.g., 0,1,2 versus 1,2,3) when rankings are used as the matrix elements.

(3) The sensitivity of the Bond Energy Algorithm to the choice of k depends on the relative magnitudes of the various matrix elements. If all the matrix elements are 0 or 1, then the ME is independent of the choice of k. If, however, the matrix elements vary greatly in magnitude, it is recommended that k be set equal to 2 instead of 1. This choice preserves scale by not overemphasizing the largest elements. For example, with k = 2, the bond strength between elements of sizes 5 and 7 will be the $\sqrt{35}$, close to their average, rather than the inflated value 35 when k = 1.

APPENDIX H

A MEASURE OF EFFECTIVENESS FOR THE MOMENT ORDERING ALGORITHM

A MEASURE OF EFFECTIVENESS FOR THE MOMENT ORDERING ALGORITHM.

It has been pointed out that one of the properties of the algorithm is to drive the array being operated upon toward a more diagonal form. This property has been utilized to define[1] a correlation coefficient, R, to measure the progress of the iterative procedure and the quality of the final result. The coefficient has been defined as follows:

$$R = \frac{S_{xy}}{S_x S_y},$$

where

$$S_x^2 = \frac{1}{T-1} \sum_{i=1}^{M} \sum_{j=1}^{N} a_{ij} \left(X_i - \overline{X}\right)^2,$$

$$S_y^2 = \frac{1}{T-1} \sum_{i=1}^{M} \sum_{j=1}^{N} a_{ij} \left(Y_j - \overline{Y}\right)^2,$$

$$S_{xy} = \frac{1}{T-1} \sum_{i=1}^{M} \sum_{j=1}^{N} a_{ij} \left(X_i - \overline{X}\right)\left(Y_j - \overline{Y}\right)$$

$$\overline{X} = \frac{1}{T} \sum_{i=1}^{M} \sum_{j=1}^{N} a_{ij} X_i$$

$$\overline{Y} = \frac{1}{T} \sum_{i=1}^{M} \sum_{j=1}^{N} a_{ij} Y_j$$

$$T = \sum_{i=1}^{M} \sum_{j=1}^{N} a_{ij}$$

$$X_i = i/M$$

$$Y_j = j/N$$

a_{ij} = element in i^{th} column and j^{th} row.

---

[1] Suggestion due to Dr. Phillip Gould of IDA.

Note that R is normalized so that its value always lies between zero and one. For the special case of a square matrix R=1 corresponds to only the main diagonal being filled, R=0 to a random distribution of value throughout the array, and R= -1 to the opposite diagonal only being filled. Initial values of R for arrays therefore are generally near zero, and as the algorithm proceeds toward a solution, R generally increases. The final value of R is a measure of the degree of diagonality obtained by the algorithm. It should be noted however, that the algorithm is *not* a direct attempt to maximize R, and that there are occasional cases in which an iteration of the algorithm will decrease R instead of increasing it.

APPENDIX I

MULTIPLE SOLUTIONS TO THE MOMENT ORDERING ALGORITHM FOR A SAMPLE 3 x 3 ARRAY

MULTIPLE SOLUTIONS TO THE MOMENT ORDERING ALGORITHM FOR A SAMPLE 3 x 3 ARRAY.

In order to investigate the factors which lead to multiple solutions to the Moment Ordering Algorithm, the following experiment was carried out. It deals with a 3 x 3 array, but it is believed that the conclusions drawn may be useful in understanding the phenomenon for the vastly more complicated cases of larger arrays.

1. A sample 3 x 3 array (Fig. I-1) was constructed. For simplicity, its rows were each nomalized to 10. Two of the rows were fixed (7,2,1 and 3,5,2), while the elements in the third were allowed to take on various values (always subject to the normalization and the restriction that all elements be non-negative).

	α	β	γ
A	7	2	1
B	3	5	2
C	X	Y	Z

11-13-69-4

FIGURE I-1. Experimental 3x3 Array

2. For every possible combination of values for the elements of the third row, the resulting array was analyzed. In particular, the number of possible stable solutions was determined.

3. The results are presented in Fig. I-2. Every point inside the triangle represents a possible third row of the array. The values of the three elements are read upwards from each face. (Note that the sum of the distance from any point inside to all three faces of the equilateral triangle is constant–in this case equal to 10.) The sets of elements corresponding to the first two rows are marked as A and B, and for each other point the multiplicity of solutions to the resulting array is shown.

4. The resulting overall pattern indicates that when the third point is colinear, or nearly so, with points A and B (that is, when the three rows are in a

well-defined order within the coordinate system of Fig. I-2) only one solution usually exists. However, in the region in the lower left hand section of the triangle, where the third point forms a triangle with A and B, rather than a straight line, three stable solutions exist. This indicates that, when one specific linear ordering exists, the algorithm will find that ordering, but that when several orderings are equally satisfactory, it may find each.

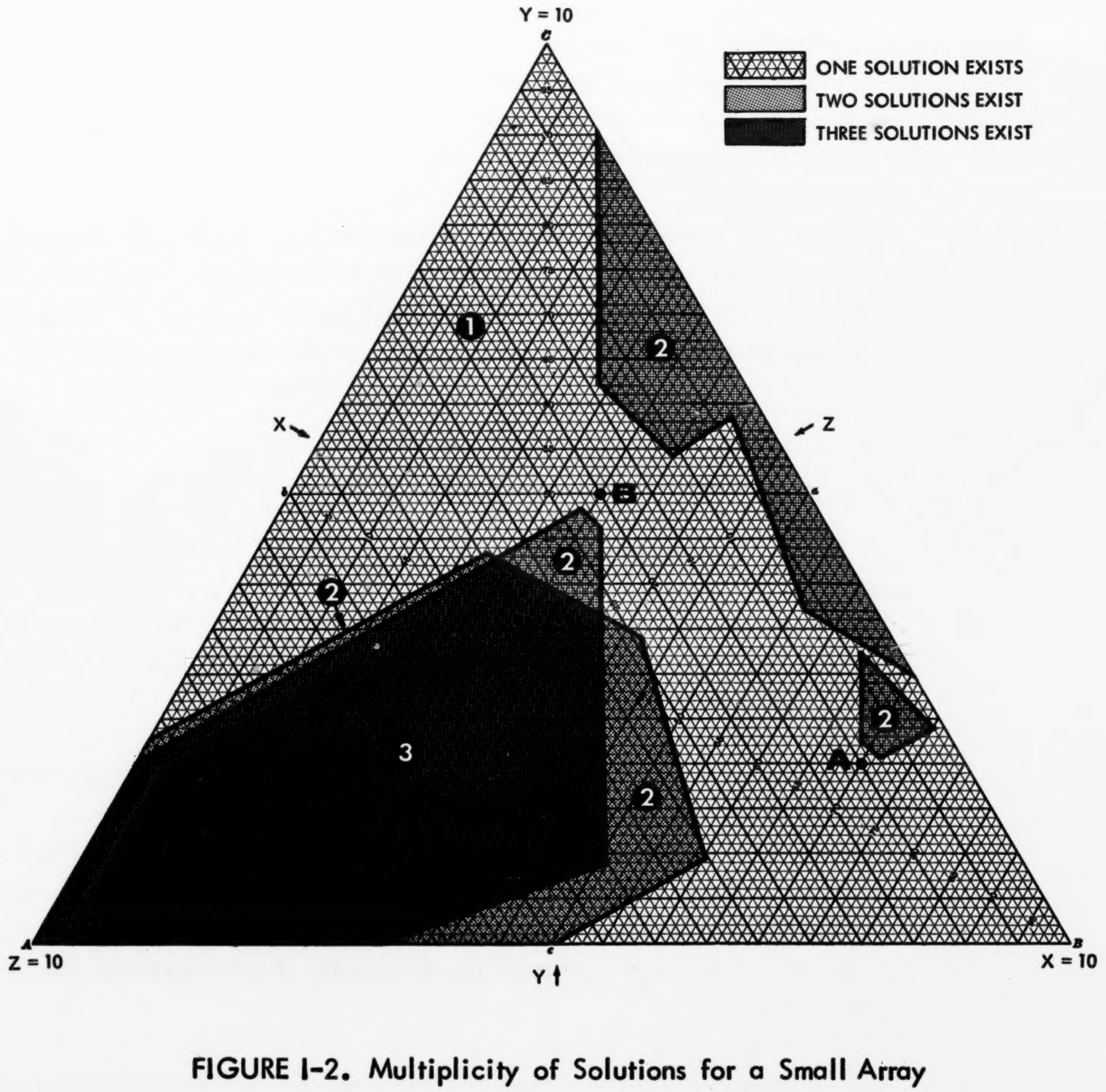


FIGURE I-2. Multiplicity of Solutions for a Small Array